UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE AUTOMAÇÃO E SISTEMAS

MÉTODOS DE OTIMIZAÇÃO: TEORIA E PRÁTICA

Versão preliminar

Eduardo Camponogara

Florianópolis, Setembro de 2006

Agradecimentos

Agradeço sinceramente o apoio recebido dos alunos Alberto X. Pavim, Agustinho Plucênio, Martín Pomar, Maurício Serra e Rodrigo Carlson na preparação da versão eletrônica das notas de aula em LaTeX.

Sumário

1 As Subáreas da Otimização **1**

1.1 Conceitos Fundamentais 1
1.1.1 Modelagem de Problemas 1
1.1.2 Elementos de um Problema de Otimização 2
1.1.3 Duas Exceções à Formulação Geral 2
1.2 Programação Linear 3
1.2.1 Problema Exemplo 3
1.3 Programação Linear Inteira 4
1.3.1 Problema Exemplo 4
1.4 Programação Quadrática 5
1.5 Mínimos Quadrados Não-Linear 6
1.6 Mínimos Quadrados (Linear) 6
1.6.1 Exemplo Ilustrativo 6
1.6.2 O Problema de Mínimos Quadrados 7
1.6.3 Ajuste de Curvas 8
1.7 Equações Não-Lineares 9
1.7.1 Aplicação em Sistemas de Controle 10
1.8 Otimização Não-Linear sem Restrições 10
1.8.1 Problema Exemplo 10
1.9 Otimização Não-Linear com Limites Superiores/Inferiores 12
1.10 Otimização Não-Linear com Restrições 12
1.10.1 Problema Exemplo 12
1.11 Programação Semi-Definida 13
1.11.1 Exemplo 13
1.12 Referências 14
1.13 Exercícios 14

2 Minimização de Funções com o Método de Descenso **17**

2.1 Problemas de Interesse 17
2.2 Fundamentos de Otimização Irrestrita 18
2.2.1 O que é uma solução para P_1? 18
2.2.2 Reconhecendo Um Mínimo Local 18
2.3 O Algoritmo de Descenso 20
2.3.1 O Algoritmo de Descenso em Detalhes 21
2.3.2 Direção de Busca 21
2.3.3 Encontrando a Direção de Busca 22

2.3.4 Encontrando o Passo . . . 22
2.3.5 Redução Suficiente e Retrocesso . . . 24
2.3.6 Convergência do Método de Descenso . . . 25
2.3.7 Taxa de Convergência do Método de Descenso Íngreme . . . 26
2.4 Referências . . . 28
2.5 Exercícios . . . 28

3 Método de Newton: Solução de Equações Não-Lineares e Minimização de Funções 31
3.1 Problemas de Interesse . . . 31
3.2 O Método de Newton em uma Variável . . . 31
3.2.1 Exemplo . . . 32
3.3 O Método de Newton para Minimização em uma Variável . . . 33
3.3.1 Exemplo . . . 33
3.4 O Método de Newton em Múltiplas Variáveis . . . 35
3.5 Minimização Irrestrita . . . 36
3.6 Convergência . . . 37
3.6.1 Convergência Linear . . . 37
3.6.2 Convergência Quadrática . . . 37
3.6.3 Convergência do Método de Newton . . . 38
3.7 Métodos de Região de Confiança . . . 39
3.8 Otimização de Funções Sujeito a Igualdades . . . 41
3.9 Referências . . . 43
3.10 Exercícios . . . 43

4 Otimização Não-Diferenciável 47
4.1 Otimização *Black-Box* . . . 47
4.2 Algoritmo Genético (AG) . . . 48
4.2.1 Genética e Evolução . . . 48
4.2.2 Adaptação Biológica . . . 48
4.2.3 Hereditariedade com Evolução Simulada . . . 49
4.2.4 Popularidade do Algoritmo Genético . . . 49
4.2.5 Algoritmo Genético em Detalhes . . . 50
4.2.6 Operador Genético *Cross-Over* . . . 51
4.2.7 Exemplo de Aplicação . . . 51
4.2.8 Questões Práticas . . . 52
4.2.9 *Schema Theorem* . . . 52
4.3 *Simulated Annealing* . . . 54
4.3.1 O Processo de *Annealing* . . . 54
4.3.2 O Algoritmo de Metropolis . . . 55
4.3.3 Exemplo: O Problema do Caixeiro Viajante . . . 57
4.4 Referências . . . 58
4.5 Exercícios . . . 59

5 Treinamento de Redes Neurais: Um Problema de Otimização **61**
5.1 Elementos Básicos das Redes Neurais 62
5.1.1 O Problema de Treinamento 62
5.1.2 ALVINN: Um Exemplo de Aplicação 63
5.1.3 Problemas Apropriados para Redes Neurais 63
5.2 Perceptron: A Primeira Unidade Neural 65
5.2.1 Treinando um Perceptron 65
5.3 Regra Delta 67
5.3.1 Treinando uma Unidade Delta 67
5.4 A Unidade *Sigmoid* 68
5.5 Referências 69
5.6 Exercícios 70

6 Programação Linear **73**
6.1 Problema Exemplo: Gerenciamento de Uma Unidade de Produção . . . 74
6.1.1 Gerente de Produção Otimista 74
6.1.2 Gerente de Produção Pessimista 74
6.2 O Problema de Programação Linear 75
6.3 Algoritmo Simplex 77
6.3.1 Exemplo 78
6.3.2 Algoritmo Simplex em detalhes 81
6.3.3 Inicialização 83
6.4 Dualidade 85
6.4.1 Motivação 85
6.4.2 O Problema Dual 86
6.4.3 O Teorema Fraco da Dualidade 87
6.4.4 O Teorema Forte da Dualidade 87
6.4.5 Folga Complementar 89
6.5 Algoritmo Simplex em Notação Matricial 90
6.5.1 Dicionário em Forma Matricial 91
6.6 Referências 92
6.7 Exercícios 92

7 Teoria dos Jogos **95**
7.1 Introdução 95
7.2 Jogos Matriciais 97
7.2.1 O Jogo da Tesoura, Pedra e Papel 97
7.2.2 Um Jogo Menos Trivial 97
7.3 Formalização 98
7.4 Estratégia Ótima para o Agente Coluna 98
7.5 Estratégia Ótima para o Agente Linha 100
7.6 Relação entre os Problemas P_x e P_y 100
7.7 Teorema Minimax 101
7.8 Jogos Quadráticos 102
7.8.1 Exemplo 103
7.9 Referências 104

7.10 Exercícios 105

8 Fluxo em Redes **107**
8.1 Dois Problemas Clássicos 107
8.1.1 O Problema de Transporte 107
8.1.2 O Problema de Alocação 109
8.2 O Problema de Fluxo Máximo 110
8.2.1 Fluxos e Cortes 112
8.2.2 Algoritmo de Caminhos Aumentantes 114
8.2.3 Implicações Combinatórias do Problema de Fluxo Máximo . . . 117
8.3 O Problema de Fluxo de Custo Mínimo 119
8.3.1 Transformações 120
8.3.2 Um Exemplo 122
8.3.3 Redes Residuais 123
8.3.4 Algoritmo de Cancelamento de Circuitos Negativos 124
8.4 Matrizes Totalmente Unimodulares 126
8.5 Referências 129
8.6 Exercícios 129

9 Linguagens de Modelagem **135**
9.1 Linguagem Mosel 135
9.1.1 Qual Interface Devemos Utilizar? 136
9.1.2 Resolvendo um Problema 136
9.1.3 Indo Mais Longe 137
9.1.4 Trabalhando com o Optimizer 138
9.1.5 Construindo um Primeiro Modelo 138
9.1.6 Usando Cadeias de Caracteres como Índices 140
9.1.7 Modelagem Versátil 141
9.2 Linguagem AMPL 142
9.2.1 Modelo AMPL do Problema da Mochila 143
9.2.2 Comentários 144
9.3 Estudo de Caso: Alocação de Rotas em Redes de Computadores 144
9.4 Formulação do Problema 144
9.4.1 Formulação em MOSEL 146
9.4.2 Formulação em AMPL 150
9.5 Referências 153
9.6 Exercícios 153

10 Fundamentos de Programação Inteira **155**
10.1 Introdução 155
10.1.1 Escalonamento de Trens 155
10.1.2 *Airline Crew Scheduling* 156
10.2 O Que É um Problema Inteiro? 156
10.2.1 Problema (Linear) Inteiro Misto 156
10.2.2 Problema (Linear) Inteiro 156
10.2.3 Problema Linear Binário 157

10.2.4 Problema de Otimização Combinatória . . . 157
10.3 Programação Linear e Arredondamento . . . 157
10.4 Formulação de PIs e PIBs . . . 158
10.4.1 Exemplo 1: Formulando o Problema de Alocação . . . 158
10.4.2 Exemplo 2: O Problema da Mochila . . . 159
10.4.3 Exemplo 3: O Problema de Cobertura por Conjuntos . . . 160
10.4.4 Exemplo 4: O Problema do Caixeiro Viajante (PCV) . . . 161
10.5 Explosão Combinatória . . . 162
10.6 Formulação de Problemas Inteiros Mistos (PIMS) . . . 163
10.6.1 Exemplo 1: Modelando Custos Fixos . . . 163
10.6.2 Exemplo 2: Localização de Depósitos sem Limites de Capacidade 164
10.6.3 Alternativas Discretas e Disjuntas . . . 165
10.7 Formulações Alternativas . . . 166
10.7.1 Formulações Equivalentes para o Problema da Mochila . . . 166
10.7.2 Localização de Depósitos sem Limites de Capacidade . . . 167
10.8 Formulações Apertadas e Ideais . . . 168
10.8.1 Formulações Equivalentes para o Problema da Mochila . . . 169
10.8.2 Formulação para o Problema de Localização de Depósitos . . . 169
10.9 Referências . . . 170
10.10Exercícios . . . 170

11 Programação Inteira: Relaxações e Algoritmo *Branch-and-Bound* 173
11.1 Condições de Otimalidade . . . 173
11.1.1 Limite Primal . . . 174
11.1.2 Limite Dual . . . 174
11.2 Relaxação Baseada em PL (Relaxação Linear) . . . 174
11.3 Relaxação Combinatória . . . 175
11.3.1 O Problema do Caixeiro Viajante . . . 175
11.3.2 O Problema da Mochila . . . 175
11.4 Relaxação Lagrangeana . . . 176
11.5 Algoritmo *Branch-and-Bound* . . . 177
11.5.1 Estratégia de Divisão e Conquista . . . 177
11.5.2 Enumeração Implícita . . . 178
11.5.3 Algoritmo *Branch-and-Bound* (B&B) . . . 181
11.6 Referências . . . 184
11.7 Exercícios . . . 184

12 Algoritmos de Planos de Corte e Desigualdades Fortes 187
12.1 Introdução a Planos de Corte . . . 187
12.2 Exemplos de Desigualdades Válidas . . . 188
12.2.1 Um conjunto 0-1 puro . . . 188
12.2.2 Um Conjunto 0-1 Misto . . . 188
12.2.3 Um Conjunto Inteiro Misto . . . 189
12.2.4 Conjunto Combinatório . . . 189
12.2.5 Arredondamento Inteiro . . . 190
12.2.6 Arredondamento Inteiro Misto . . . 190

12.3 Teoria de Desigualdades Válidas 190
12.3.1 Desigualdades Válidas para Problemas Lineares 191
12.3.2 Desigualdades Válidas para Problemas Inteiros 191
12.3.3 Procedimento Chvátal-Gomory para Geração de Desigualdades Válidas 192
12.4 O Algoritmo de Planos de Corte 192
12.5 Algoritmo de Planos de Corte Usando Cortes de Gomory Fracionários . 193
12.5.1 Exemplo 194
12.6 Desigualdades Disjuntivas 195
12.6.1 Exemplo 195
12.6.2 Desigualdades Disjuntivas para Problemas 0-1 196
12.7 Exercícios 197

13 Programação com Restrições Lógicas 199

14 Programação Dinâmica: Domínio Discreto 201
14.1 Um Exemplo de Programação Dinâmica 202
14.1.1 Calculando Números de Fibonacci 203
14.2 Programação Dinâmica para o Problema da Mochila 204
14.2.1 Complexidade do Algoritmo 205
14.3 Elementos de um Algoritmo DP: Sequência Crescente Mais Longa . . . 206
14.3.1 Construindo um Algoritmo 207
14.4 Edição Automática de Cadeias de Símbolos ("Approximate String Matching") 208
14.4.1 Projeto de um Algoritmo DP 208
14.5 Exercícios 210

15 Programação Dinâmica: Domínio Contínuo 215
15.1 Introdução 215
15.1.1 Exemplo: Controle de Inventário 216
15.1.2 Distinção Entre Otimização "Open-Loop" e "Closed-Loop" . . . 217
15.2 O Problema Básico de Programação Dinâmica 218
15.2.1 O Valor da Informação 218
15.3 O Algoritmo de Programação Dinâmica 219
15.3.1 Princípio da Otimalidade 219
15.3.2 O Algoritmo de Programação Dinâmica 219
15.4 Sistemas Lineares com Custo Quadrático 220
15.4.1 Aplicando o Algoritmo de Programação Dinâmica 221
15.4.2 A Equação de Riccati e seu Comportamento Assintótico 222
15.5 Exercícios 224

16 Programação Não-Linear Restrita: Fundamentos e Condições de Otimalidade 229
16.1 Teoria da Otimização Não-Linear sob Restrições 229
16.1.1 Solução Local × Solução Global 230
16.1.2 Suavidade das Funções 230

16.2 Exemplos . . . 231
16.2.1 Uma Restrição de Igualdade . . . 232
16.2.2 Uma Restrição de Desigualdade . . . 233
16.2.3 Duas Desigualdades . . . 235
16.3 Condições de Otimalidade de Primeira Ordem . . . 238
16.3.1 Exemplo . . . 240
16.4 Exercícios . . . 241

17 Programação Não-Linear: Fundamentos de Algoritmos 243
17.1 Categorizando Algoritmos de Otimização . . . 244
17.1.1 O Método da Função Penalidade . . . 244
17.1.2 O Método de Barreiras . . . 244
17.1.3 O Método Lagrangeano Aumentado . . . 245
17.1.4 Método Seqüencial Linear . . . 245
17.1.5 Programação Quadrática Seqüencial . . . 245
17.2 Eliminação de Variáveis . . . 245
17.2.1 Exemplo: Problemas Que Podem Surgir . . . 246
17.2.2 Eliminação de Restrições Lineares . . . 246
17.2.3 Os Efeitos das Desigualdades . . . 248

18 Programação Quadrática 251
18.1 Exemplo: Otimização de Portofolio . . . 252
18.2 Propriedades de Problemas Quadráticos com Apenas Restrições de Igualdade . . . 253
18.2.1 Resolvendo o Sistema KKT . . . 255
18.3 Problemas Quadráticos Sob Restrições . . . 255
18.3.1 Condições de Otimalidade . . . 256
18.4 O Método de Conjunto Ativo para Problemas Quadráticos Convexos . 256
18.4.1 Exemplo . . . 259
18.5 Exercícios . . . 261

19 Programação Não-Linear sob Restrições: Algoritmos 265
19.1 Algoritmos . . . 265
19.2 O Método de Barreira Logarítmica . . . 265
19.2.1 Propriedades das Funções de Barreira Logarítmica . . . 266
19.2.2 Algoritmo Baseado na Função Barreira Logarítmica . . . 268
19.2.3 Propriedades da Função Log-Barrier e Algoritmo Geral . . . 269
19.2.4 Manipulando Restrições de Igualdade . . . 270
19.3 Método Lagrangeano Aumentado . . . 271
19.3.1 Motivação e Estrutura do Algoritmo . . . 271
19.3.2 Exemplo . . . 272
19.3.3 Extensão para Restrições de Desigualdade . . . 273
19.3.4 Propriedades do Lagrangeano Aumentado . . . 275
19.4 Programação Linear Sequencial . . . 275
19.5 Programação Quadrática Sequencial . . . 276
19.5.1 O método SQP Local . . . 276

19.5.2 Linhas Gerais do Método SQP 278
19.5.3 Restrições de desigualdade 279
19.5.4 Implementação de SQP 280
19.6 Exercícios 280

A Exercícios Resolvidos **287**
A.1 Introdução à Otimização 288
A.2 Minimização de Funções com o Método de Descenso 290
A.3 Método de Newton 294
A.4 Otimização Black-Box 298
A.5 Treinamento de Redes Neurais 299
A.6 Programação Linear 300
A.7 Teoria dos Jogos 305
A.8 Fluxo em Redes 308
A.9 Linguagens de Modelagem 312
A.10 Programação Inteira: Fundamentos 315
A.11 Programação Inteira: Relaxações e Algoritmo *Branch-and-Bound* . . . 320
A.12 Algoritmos de Planos de Corte e Desigualdades Fortes 328
A.13 Programação Lógica 330
A.14 Programação Dinâmica: Domínio Discreto 331
A.15 Programação Dinâmica: Domínio Contínuo 339
A.16 Programação Não-Linear Sob Restrições: Fundamentos 343
A.17 Programação Não-Linear Sob Restrições: Fundamentos de Algoritmos . 345
A.18 Programação Quadrática 346
A.19 Programação Não-Linear sob Restrições: Algoritmos 349

Lista de Figuras

1.1 Predição peso com base na altura 8
1.2 Instância do problema de localização de central telefônica. 11

2.1 Exemplo de função com um número infinito de ótimos locais 18
2.2 Ilustração do gradiente de uma função, $\nabla f(x_k)$, que assume a direção ortogonal à curva de nível que passa por x_k. 21
2.3 Ilustração de passo satisfatório. 23
2.4 Ilustração das condições de Armijo e de curvatura. 23

3.1 Ilustração do processo iterativo de Newton. 34
3.2 Função $\cos(x)$ 34
3.3 Modelo mecânico. O modelo consiste de um tabuleiro com um furo perfurado a cada ponto y_k. Através da cada buraco, um fio é passado com o correspondente peso w_k preso a sua ponta. Os outros extremos dos fios são amarrados com um nó. Na ausência de fricção e fios entrelaçados, as forças atingem um equilíbrio no ponto do nó quando este está localizado na solução ótima x^*. 45

4.1 Operador *cross-over* de um ponto. 51
4.2 Operador *cross-over* de dois pontos. 51
4.3 Fenômeno de *crowding*. 53
4.4 Empacotamento de componentes eletrônicos em circuitos integrados. . . 54
4.5 Ilustração do processo de *annealing*. 55
4.6 Comportamento típico do nível de energia conforme processo de *annealing*. 56
4.7 Probabilidade de transição com a queda de temperatura. 56
4.8 Exemplo de rota para uma instância particular. 57
4.9 Ilustração dos operadores de perturbação 58

5.1 Exemplo de topologia de redes neurais. 61
5.2 Unidade de processamento neural. 62
5.3 Rede neural com camadas intermediárias. 63
5.4 Rede neural do sistema ALVINN. 64
5.5 Perceptron. 65
5.6 Conjuntos separáveis e não separáveis linearmente. 66
5.7 Unidade delta. 67
5.8 Função *sigmoid*. 69
5.9 Sistema neural para reconhecimento de palavras 69
5.10 Topologia da rede neural. 71

6.1 Exemplo de problema infactível. As regiões achuradas indicam as regiões factíveis para cada uma das restrições. Observe que a interseção das regiões achuradas é vazia. 77
6.2 Exemplo de problema ilimitado. Podemos avançar dentro da região factível de maneira a crescer o valor da função objetivo sem limites. Por exemplo, ao longo da direção $(x_1, 0)$ podemos avançar sem limites, fazendo o valor da função objetivo aumentar. 78
6.3 Método simplex como um processo iterativo. 91

7.1 Curvas de nível, conjuntos reativos, solução Nash e pontos Pareto de um jogo quadrático entre dois agentes. 104

8.1 Problema de transporte. 108
8.2 Alocação balanceada de tarefas em dois processadores como um problema de corte mínimo. A alocação ótima consiste em processar os módulos 1, 2 e 4 no processador 1, deixando os módulos 3 e 5 no processador 2. O custo de processamento dos módulos é igual 15 enquanto que o custo de comunicação é 3, perfazendo um custo total de 18 unidades. O custo total é a capacidade do corte s-t mínimo como indicado na figura. 112
8.3 Exemplo de rede residual $G(x)$ obtida a partir de uma rede G e fluxo x. 113
8.4 Rede residual $G(x^0)$ para fluxo x^0 nulo. 115
8.5 Rede residual $G(x^1)$ para fluxo x^1. 115
8.6 Rede residual $G(x^2)$ para fluxo x^2. 116
8.7 Rede residual $G(x^3)$ para fluxo x^3. 116
8.8 Transformação de arestas em arcos. 121
8.9 Remoção de limite inferior nos arcos. 121
8.10 Eliminando capacidade de arcos. 122
8.11 Exemplo de problema de fluxo de custo mínimo. As linhas tracejadas correspondem ao fluxo ótimo. 122
8.12 Exemplo de rede residual. 124
8.13 Exemplo de rede residual. 125
8.14 Primeira iteração do algoritmo de cancelamento de circuito negativo. A figura mostra a rede residual $G(x^0)$, o circuito com custo negativo em linhas tracejadas e o fluxo x^0. O circuito é dado por $w^0 = ((4,2),(2,3),(3,4))$ que tem custo $c(w^0) = -1$ e capacidade máxima $\delta(w^0) = 2$. 127
8.15 Segunda iteração do algoritmo de cancelamento de circuito negativo. A figura mostra a rede residual $G(x^1)$, o ciclo $w^1 = ((3,2),(2,1),(1,3))$ com custo $c(w^1) = -1$ e capacidade máxima $\delta(w^1) = 1$, e o fluxo x^1. . . 132
8.16 Terceira iteração do algoritmo de cancelamento de circuito negativo. Na figura é dada a rede residual $G(x^2)$ e o respectivo fluxo x^2. Também é indicado o circuito $w^2 = ((2,3),(3,4),(4,2))$ com custo $c(w^2) = -3$ e capacidade $\delta(w^2) = 1$. 132
8.17 Fluxo ótimo x^3 e rede residual $G(x^3)$. Não há ciclo de custo negativo. . 133

8.18 Propriedade de unimodularidade das matrizes de incidência de grafos. Um arco sem nó de origem indica fluxo injetado no nó destino, enquanto que um arco sem nó de destino indica fluxo consumido pelo nó origem. 133
8.19 Grafo $G = (V, A)$ com a especificação do problema de fluxo em rede. . . 134

9.1 Rede exemplo do problema de alocação de rotas. 150

10.1 Ilustração da solução obtida através de arredondamento 158
10.2 Ilustração da restrição de conectividade 162
10.3 Função com custo fixo 163
10.4 Região factível não convexa 165
10.5 Formulações equivalentes alternativas 167
10.6 Formulações alternativas 168
10.7 Formulações apertadas 169
10.8 Grafo da instância do problema do caixeiro viajante 171
10.9 Rede de abastecimento. 172

11.1 Árvore de enumeração 178
11.2 Árvore de enumeração 178
11.3 Árvore de enumeração completa, explícita 179
11.4 Eliminação por otimalidade 179
11.5 Eliminação por limite 180
11.6 Nenhum ramo da árvore pode ser eliminado 180
11.7 Quebra do primeiro nó da árvore de enumeração B&B 182
11.8 Dividindo S_1 em S_{11} e S_{12} 182
11.9 Cortando o nó S_{12} por meio da condição de otimalidade 183

12.1 Espaço de soluções factíveis e desigualdade válida $x \leqslant 14 - 4(2 - y)$. 189
12.2 Desigualdades disjuntivas 196

14.1 Árvore de recursão do algoritmo recursivo para cálculo do número de Fibonacci 204
14.2 Valores $D[i, j]$ calculados pelo algoritmo DP ao calcular a distância de edição mínima entre o padrão $P = abcdefghijkl$ e o texto $T = bcdefffghixkl$ 211
14.3 Hierarquia da empresa XYZ 213

15.1 Estrutura do sistema linear com controle ótimo de realimentação . . . 222
15.2 Sistema de fornos para aquecimento de materiais 224
15.3 Pêndulo invertido 227

16.1 Fronteiras não suaves 231
16.2 Espaço de soluções 232
16.3 Direções de descenso 233
16.4 Restrição de desigualdade 234
16.5 Região de descenso não vazia 235
16.6 Região de descenso vazia 236
16.7 Meio disco 236

16.8 Região de descenso vazia 237
16.9 Ambas as restrições 238
16.10Região factível 240

17.1 Curvas de nível das funções $x^2 + y^2$ e $y^2 = (x-1)^3$ 247

18.1 Região factível e curvas de nível da função objetivo. 260
18.2 Seqüência de iterandos. 261

19.1 Curvas de nível da função $P(x, \mu) = x - \mu \log x - \mu \log(1-x)$ para $\mu \in \{0.01, 0.1, 0.4, 1\}$ 267
19.2 Curvas de nível da função $P(x, \mu) = (x_1 + \frac{1}{2})^2 + (x_2 - \frac{1}{2})^2 - \mu\,(\log x_1 + \log(1-x_1) + \log x_2 + \log(1-x_2))$ para $\mu \in \{22, 5, 2\}$ 268
19.3 Curvas de nível da função $\mathcal{L}_A(x, -0.4, 50)$ 273
19.4 Elipsóide 282

A.1 Soluções do sistema de equações não-lineares 295
A.2 Exemplo de problema de fluxo em redes 309
A.3 Ilustração da redução do problema de caminhos mínimos com pedágio ao problema de fluxo de custo mínimo através da operação de quebra de vértices. 311
A.4 Rede de abastecimento. 319
A.5 Árvore *branch-and-bound* parcial 320
A.6 Árvore *branch-and-bound* parcial 321
A.7 Árvore *branch-and-bound* parcial 321
A.8 Árvore *branch-and-bound* parcial 322
A.9 Árvore *branch-and-bound* parcial 323
A.10 Árvore *branch-and-bound* parcial 324
A.11 Árvore *branch-and-bound* parcial 325
A.12 Árvore *branch-and-bound* parcial 326
A.13 Exemplo de execução do algoritmo de programação dinâmica para subsequência mais longa de duas cadeias de caracteres. 335
A.14 Grafo para cômputo de todos os possíveis emparelhamentos de subcadeias mais longas. 336
A.15 Processo Markoviando do jogo par-ímpar. 342
A.16 Trajetória de iterandos produzida pelo algoritmo de barreiras. 350

Lista de Tabelas

1.1 Dados amostrais 7

3.1 Iterações do método de Newton. 33

4.1 População inicial 52

5.1 Amostras para treinamento da rede neural 70

8.1 Capacidade de produção dos fornecedores 109
8.2 Demanda dos clientes 109
8.3 Custo unitário de transporte 109
8.4 Custo de processamento 111
8.5 Custo de comunicação 111

9.1 Dados do problema da mochila 139
9.2 Sequenciamento de tarefas 154

10.1 Número de soluções em função do tamanho do problema 162
10.2 Crescimento de funções 162

14.1 Tabelas DP para o Problema da Mochila 206
14.2 Tabelas DP para o Problema de Subseqüência Mais Longa 208

A.1 Tabela $D[i, j]$ para emparelhamento perfeito 332
A.2 Tabela $D[i, j]$ para emparelhamento flutuante 332
A.3 Resolução das recorrências do algoritmo de programação dinâmica . . . 337
A.4 Jogos dos palitos 338
A.5 Jogos dos palitos 338

Notação

Capítulo 1

As Subáreas da Otimização

Otimização é a área da Matemática Aplicada que se preocupa em calcular e computar valores ótimos para variáveis de decisão que induzem desempenho ótimo, ao mesmo tempo que satisfazem restrições, de um modelo matemático.

1.1 Conceitos Fundamentais

1.1.1 Modelagem de Problemas

A representação da realidade é uma necessidade da sociedade moderna, seja pela impossibilidade de lidar diretamente com a realidade, seja por aspectos econômicos, seja pela complexidade. Assim, busca-se a representação da realidade por meio de modelos que sejam bem estruturados e representativos desta realidade.

> *Modelos são representações simplificadas da realidade que preservam, para determinadas situações e enfoques, uma equivalência adequada.*

Dentre as características de um modelo se destacam a sua capacidade representativa e a capacidade de simplificação da realidade. A capacidade representativa de um modelo deve ser validada por meio de experimentação, análise numérica, ensaios ou qualquer outro método que verifique a acurácia das predições obtidas com o modelo. A modelagem de um problema complexo não é uma tarefa trivial. Invariavelmente depende de fatores subjetivos como intuição, experiência, criatividade e poder de síntese. A formulação consiste em traduzir o modelo em uma linguagem formal, normalmente expressa em notação matemática e compreendendo variáveis, equações, desigualdades e fórmulas. Os processos de formulação e validação são *iterativos*, pois envolvem múltiplas etapas de tentativa e erro, e *interativos* à medida que o modelador deve intervir continuamente objetivando refinar o modelo.

Uma abordagem freqüentemente empregada na formulação e resolução de problemas consiste no emprego de modelos de otimização, os quais visam maximizar (minimizar) um critério de desempenho como, por exemplo, a produção de um dado insumo, sujeito a restrições que descrevem as condições operacionais. A linguagem utilizada pela otimização para expressar os problemas de uma forma declarativa é conhecida universalmente por *programação matemática*. No que segue, apresentamos a linguagem

de programação matemática e exemplificamos o seu uso em uma série de problemas ilustrativos e de interesse geral.

1.1.2 Elementos de um Problema de Otimização

Tipicamente, um problema de otimização tem três elementos constituintes:

1) Variáveis de Decisão: Parâmetros cujos valores definem uma solução para o problema. Em um sistema de produção, esses parâmetros podem definir as quantidades produzidas e os recursos utilizados.

2) Função Objetivo: Uma função das variáveis de decisão a ser minimizada ou maximizada. No sistema de manufatura, podemos estar interessados em minimizar custos, reduzir o número de homens-hora e conseqüentemente aumentar a produtividade.

3) Restrições: Um conjunto de funções que define o espaço factível de soluções. No sistema de manufatura, as restrições estabelecem limites para os recursos utilizados, restrições operacionais do processo de produção, bem como limitações físicas e tecnológicas.

Formulação do Problema de Otimização Generalizado

O problema geral de otimização é expresso em programação matemática como:

$$\begin{array}{ll} \text{Minimize} & f(x) \\ \text{Sujeito a :} & \\ & g(x) \leqslant 0 \\ & h(x) = 0 \\ & x \in \mathbb{R}^n \end{array} \tag{1.1}$$

onde: $f : \mathbb{R}^n \to \mathbb{R}$ é a função objetivo; $g : \mathbb{R}^n \to \mathbb{R}^p$ e $h : \mathbb{R}^n \to \mathbb{R}^q$ são restrições que limitam o espaço de soluções factíveis; e x é o vetor com as variáveis de decisão.

1.1.3 Duas Exceções à Formulação Geral

São duas as classes principais de exceções à formulação geral do problema de otimização.

1) Problemas sem função objetivo: O usuário deseja apenas encontrar um conjunto de decisões que sejam viáveis, isto é, encontre $x \in \mathbb{R}^n$ tal que:

$$\begin{aligned} g(x) &\leqslant 0 \\ h(x) &= 0 \end{aligned}$$

2) Problemas com múltiplos objetivos: Em problemas reais não é incomum procurar otimizar mais do que um objetivo. No problema de manufatura o usuário pode desejar maximizar o lucro, maximizar a qualidade dos itens manufaturados e ainda minimizar o tempo de produção. Usualmente, estes problemas são

reduzidos a problemas envolvendo apenas um objetivo (combinando-se múltiplos objetivos em apenas um ou, alternativamente, escolhendo-se um objetivo e introduzindo restrições). Tais problemas são transcritos em programação matemática como:

$$\begin{array}{rl} \text{Minimize} & f_1(x) \\ \vdots & \\ \text{Minimize} & f_k(x) \\ x \in \mathbb{R}^n & \\ \text{Sujeito a :} & g(x) \leqslant 0 \\ & h(x) = 0 \end{array}$$

1.2 Programação Linear

O problema de programa linear é um caso particular de (1.1) cuja função objetivo e restrições são todas lineares. Matematicamente, $f(x) = c^T x$, $g(x) = Ax - a$ e $h(x) = Bx - b$ sendo c, a e b vetores e A e B matrizes com dimensões apropriadas. O problema e os algoritmos de programação linear são amplamente empregados tendo inúmeras aplicações. No que segue apresentamos uma aplicação e a formulação geral.

1.2.1 Problema Exemplo

Um atleta deseja encontrar uma dieta otimizada, ou seja, um programa alimentar com tipos e quantidades de alimentos que atendam às suas necessidades mínimas. Os alimentos devem ser escolhidos de forma a minimizar o preço total. Os dados do problema são:

- N alimentos, tais como arroz, feijão, alface, etc;
- M tipos de substâncias alimentares, como proteínas, lipídios, etc;
- c_n é o preço unitário do alimento n;
- $a_{m,n}$ é a quantidade de substância m contida em cada unidade de alimento n; e
- b_m é a quantidade mínima de substância m a ser ingerida pelo atleta.

Exercício: modele o problema em programação matemática.

1) Variáveis: x_n é quantidade de alimento n a ser comprada e ingerida, $n = 1, \ldots, N$.

2) Restrições:

$$\left\{ \begin{array}{ccccccccc} a_{1,1}x_1 & + & a_{1,2}x_2 & + & \ldots & + & a_{1,N}x_N & \geqslant & b_1 \\ a_{2,1}x_1 & + & a_{2,2}x_2 & + & \ldots & + & a_{2,N}x_N & \geqslant & b_2 \\ \vdots & & \vdots & & \ldots & & \vdots & & \vdots \\ a_{M,1}x_1 & + & a_{M,2}x_2 & + & \ldots & + & a_{M,N}x_N & \geqslant & b_M \end{array} \right.$$

onde $x_1, x_2, \ldots, x_N \geqslant 0$.

3) Função objetivo: $f = c_1x_1 + c_2x_2 + \ldots + c_Nx_N$

4) Formulação compacta:

$$\begin{array}{ll} \text{Minimize} & c^T x \\ x \in \mathbb{R}^N & \\ \text{Sujeito a :} & Ax \geq b \\ & x \geq 0 \end{array} \tag{1.2}$$

1.3 Programação Linear Inteira

Uma variedade de problemas reais podem ser formulados com o emprego de variáveis discretas. Dentre eles, citamos o problema de agendamento de trens, o problema de agendamento de tripulações de aviões e o problema de alocação em sistemas de telecomunicações. Todos estes problemas têm uma característica em comum: fazem uso de variáveis inteiras ou discretas. Não se pode por exemplo dividir um vagão de trem em frações; não se pode alocar meio piloto a uma aeronave; e não se pode instalar uma fração de um servidor de telecomunicações. Estes e muitos outros problemas fazem parte do universo da programação linear inteira, que engloba problemas da forma:

$$\begin{array}{ll} \text{Minimize} & c^T x \\ \text{Sujeito a :} & \\ & Ax \geq b \\ & Cx = d \\ & x \geq 0 \\ & x \in \mathbb{Z}^n \end{array}$$

1.3.1 Problema Exemplo

Aqui ilustramos uma aplicação do problema de programação inteira que envolve a instalação de depósitos e unidades produtoras de aço que venham a suprir as demandas de clientes, ao mesmo tempo que os custos de instalação e transportes sejam minimizados. Temos que decidir, dentre um conjunto de locais pré-selecionados, onde instalar as unidades produtoras e quais unidades instaladas serão responsáveis pela demanda de cada cliente.

1) Dados do Problema: os dados abaixo constituem uma instância do problema:

i. um número m de possíveis locais para instalação de depósitos e siderúrgicas;

ii. um número n de clientes;

iii. d_i é a demanda de aço do cliente i e esta deve ser suprida por precisamente um depósito ou siderúrgica;

iv. u_j é a capacidade de um possível depósito a ser instalado no local j;

v. o custo de transporte do depósito j para o cliente i é $c_{i,j}$; e

vi. o custo de instalação do depósito j é f_j.

2) Tarefa: Formule o problema de definir quais depósitos devem ser instalados de maneira a suprir a demanda e, ao mesmo tempo, minimizar o custo total de instalação e transporte.

3) Variáveis: as variáveis de decisão são:

i. $x_{ij} = 1$ se o cliente i é atendido pelo depósito j, $x_{ij} = 0$ caso contrário; e

ii. $y_j = 1$ se o depósito j é instalado, $y_j = 0$ caso contrário.

4) Formulação:

$$
\begin{array}{lll}
\text{Minimize} & \displaystyle\sum_{i=1}^{n}\sum_{j=1}^{m} c_{ij}x_{ij} + \sum_{j=1}^{m} f_j y_j & \\
\text{Sujeito a :} & & \\
& \displaystyle\sum_{i=1}^{n} d_i x_{ij} \leqslant u_j y_j & j = 1, \ldots, m \\
& \displaystyle\sum_{j=1}^{m} x_{ij} = 1 & i = 1, \ldots, n \\
& x_{ij} \in \mathbb{B} & i = 1, \ldots, n \\
& & j = 1, \ldots, m \\
& y_j \in \mathbb{B} & j = 1, \ldots, m
\end{array}
$$

sendo $\mathbb{B} = \{0, 1\}$.

1.4 Programação Quadrática

O problema geral de programação quadrática é formulado como segue:

$$
\begin{array}{lll}
\text{Minimize} & \frac{1}{2}x^T Q x + c^T x & \\
\text{Sujeito a :} & & \\
& Ax \leqslant b & \\
& Cx = d &
\end{array}
$$

onde Q é uma matriz simétrica.

Programação quadrática tem aplicações em identificação de parâmetros para modelos de processos, modelos estruturais e sistemas de controle, e em algoritmos como SQP (*Sequential Quadratic Programming*).

A dificuldade de se resolver tais problemas depende da natureza da matriz Q. Quais características de Q tornam o problema difícil? Se $Q \geqslant 0$ (*positiva semi-definida*)[1] ou $Q > 0$ (*positiva definida*) o problema é relativamente fácil de ser resolvido (ou seja, encontrar a solução ótima global). Se Q é indefinida (ou negativa semi-definida ou definida) então o problema é muito difícil.

[1] $Q \in \mathbb{R}^{n\times n}$ é dita positiva semi-definida se $x^T Q x \geqslant 0$ para todo $x \in \mathbb{R}^n$. A matriz é dita positiva definida se a desigualdade é estrita para todo $x \neq 0$. Se $Q = Q^T$, então Q é positiva semi-definida se e somente se os autovalores de Q são não negativos. Note que os autovalores de matrizes simétricas são números reais. A matriz Q é dita indefinida se existe $x, y \in \mathbb{R}^n$ tal que $x^T Q x > 0$ e $y^T Q y < 0$.

1.5 Mínimos Quadrados Não-Linear

O problema dos mínimos quadrados não-linear consiste de um problema da seguinte forma:

$$\begin{array}{ll}\text{Minimize} & \frac{1}{2}\|f(x)\|^2 \\ x \in \mathbb{R}^n & \end{array}$$

onde $\| \bullet \|$ corresponde à norma Euclidiana e $f(x) : \mathbb{R}^n \to \mathbb{R}^m$ é uma função qualquer, contínua e diferenciável. Tais problemas têm aplicações no casamento de modelos com dados experimentais, tipicamente encontrados em estudos econômicos, aprendizagem automática e engenharia.

Seja $\{(y_k, z_k) : k = 1, \ldots, K\}$ um conjunto de pares entrada-saída de uma função $z = h(y)$ desconhecida, $y_k \in \mathbb{R}^p$ e $z_k \in \mathbb{R}^q$. Suponha que uma função $g_x(y)$ com parâmetros dado pelo vetor $x \in \mathbb{R}^n$ é sugerida como aproximação de $h(y)$. Existe, portanto, uma família $\mathcal{F} = \{g_x : x \in \mathbb{R}^n\}$ de aproximadores. O problema então encontrar a função g_x que minimiza o erro de aproximação:

$$e = \sum_{k=1}^{K} (g_x(y_k) - z_k)^2$$

Definindo $f(x) = [f_1(x), \ldots, f_K(x)]$ tal que $f_k(x) = g_x(y_k) - z_k$, temos que o problema de encontrar o aproximador de menor erro se reduz a um problema de mínimos quadrados não-linear.

Uma aplicação do problema de mínimos quadrados não-linear é o treinamento de redes neurais. Dado uma série de exemplos de treinamento (pares entrada-saída), o problema de treinar uma rede neural a aproximar a função descrita pelos pares entrada-saída se reduz a um problema de mínimos quadrados não-linear.

1.6 Mínimos Quadrados (Linear)

Vários problemas encontrados na prática como, por exemplo, o problema de ajuste de curvas e de identificação de sistemas, podem ser expressos como um problema de encontrar os parâmetros de uma função linear nos parâmetros que aproxime os dados observados.

Abaixo ilustramos uma aplicação do problema de mínimos quadrados (linear) ao problema de encontrar uma função de predição do peso de pessoas em função da altura com base em exemplos amostrais. Na seqüência formalizamos o problema de mínimos quadrados e finalizamos com uma aplicação ao problema de ajuste de curvas generalizado.

1.6.1 Exemplo Ilustrativo

Seja $w(h)$ um modelo que descreve a relação entre a altura e o peso médio das pessoas do sexo feminino. Suponha que o modelo escolhido é um polinômio da forma:

$$w(h) = x_3 h^3 + x_2 h^2 + x_1 h + x_0,$$

ou seja, $w(h)$ é um polinômio de terceira ordem que modela o peso como uma função da altura. Os dados amostrais são dados na Tabela 1.1

Tabela 1.1: Dados amostrais

Amostra (i)	Altura (h_i)	Peso (w_i)
1	1.50	55
2	1.54	53
3	1.58	56
4	1.60	52
5	1.65	58
6	1.67	59
7	1.70	64
8	1.72	71
9	1.72	71
10	1.75	75
11	1.80	82
12	1.81	81

Problema Exemplo

Encontre os parâmetros x_3, x_2, x_1 e x_0 que minimizam a função erro, a qual consiste na soma dos quadrados dos erros de predição:

$$\begin{array}{ll} \text{Minimize} & \frac{1}{2}\sum_{i=1}^{n} \|w(h_i) - w_i\|^2 \\ x_0, x_1, x_2, x_3 & \end{array}$$

A solução ótima para o problema acima, tomando como dados as entradas da Tabela 1.1, é:

$$\begin{aligned} x_0 &= 100.0000 \\ x_1 &= -32.0025 \\ x_2 &= -64.3704 \\ x_3 &= 42.1669 \end{aligned}$$

A Figura 1.1 ilustra os dados amostrais juntamente com a curva de aproximação $w(h)$ dada pelo polinômio $w(h) = x_3h^3 + x_2h^2 + x_1h + x_0$.

1.6.2 O Problema de Mínimos Quadrados

Consideremos inicialmente um sistema linear com mais equações do que variáveis:

$$\left\{\begin{array}{ccccccccc} a_{11}x_1 & + & a_{12}x_2 & + & \ldots & + & a_{1n}x_n & = & b_1 \\ a_{21}x_1 & + & a_{22}x_2 & + & \ldots & + & a_{2n}x_n & = & b_2 \\ a_{31}x_1 & + & a_{32}x_2 & + & \ldots & + & a_{3n}x_n & = & b_3 \\ \vdots & & & & & & & & \\ a_{m1}x_1 & + & a_{m2}x_2 & + & \ldots & + & a_{mn}x_n & = & b_m \end{array}\right.$$

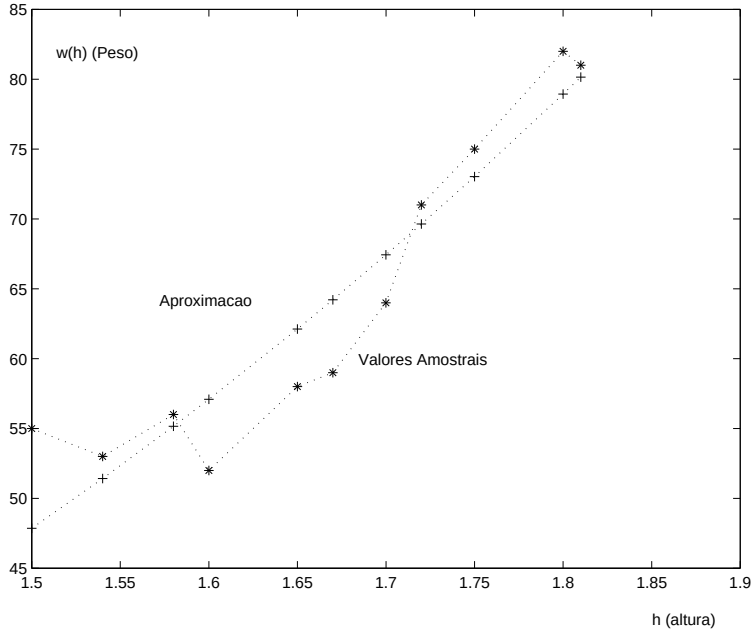


Figura 1.1: Predição peso com base na altura

o qual pode ser colocado na forma mais compacta

$$Ax = b$$

onde $A \in \mathbb{R}^{m \times n}$, $x \in \mathbb{R}^{n \times 1}$ e $b \in \mathbb{R}^{m \times 1}$. Vamos assumir que não existe solução x tal que $Ax = b$, o que ocorre quando $posto(A) < posto([A|b])$.

O problema de encontrar uma solução aproximada para $Ax = b$ pode ser colocado em programação matemática como um problema de mínimos quadrados linear:

$$P_{MQ}: \quad \begin{array}{l} \text{Minimize} \quad \|Ax - b\| \\ x \in \mathbb{R}^n \end{array}$$

onde $r = Ax - b$ é chamado de resíduo. A solução x para P_{MQ} com menor erro residual $\|r\|$ é dita solução de mínimos quadrados. O problema P_{MQ} pode ser colocado na forma equivalente P'_{MQ}:

$$P'_{MQ}: \quad \begin{array}{l} \text{Minimize} \quad \|Ax - b\|^2 = (Ax - b)^T(Ax - b) = \sum_{i=1}^{m}(a_i^T x - b_i)^2 \\ x \in \mathbb{R}^n \end{array}$$

onde a_i^T é a i-ésima linha de A.

1.6.3 Ajuste de Curvas

O problema geral de ajuste de curvas pode ser colocado como segue. Ajuste a curva dada pela função

$$g(t) = x_1 g_1(t) + x_2 g_2(t) + \ldots + x_n g_n(t)$$

aos dados $(t_1, y_1), \ldots, (t_m, y_m)$, ou seja, gostaríamos que:

$$\begin{aligned} g(t_1) &= y_1 \\ g(t_2) &= y_2 \\ \vdots &= \vdots \\ g(t_m) &= y_m \end{aligned}$$

onde $g_i(t) : \mathbb{R} \to \mathbb{R}$, $i = 1, \ldots, n$, são funções quaisquer mas conhecidas; tais funções são ditas bases. Note que as variáveis do problema são $x_1, x_2, \ldots, x_n$ e tipicamente $m >> n$.

Podemos colocar o problema de ajuste de curvas como um problema mínimos quadrados, o qual assume a forma:

$$\begin{aligned} \text{Minimize} \quad f &= \sum_{i=1}^{m} [g(t_i) - y_i]^2 \\ &= \sum_{i=1}^{m} [x_1 g_1(t) + x_2 g_2(t) + \ldots + x_n g_n(t) - y_i]^2 \end{aligned}$$

que, por sua vez, pode ser expresso em forma matricial:

$$\begin{aligned} \text{Minimize} \quad & \|Ax - b\|^2 \\ x \in \mathbb{R}^n & \end{aligned}$$

onde:

$$A = \begin{bmatrix} g_1(t_1) & g_2(t_1) & \ldots & g_n(t_1) \\ g_1(t_2) & g_2(t_2) & \ldots & g_n(t_2) \\ \vdots & \vdots & \vdots & \vdots \\ g_1(t_m) & g_2(t_m) & \ldots & g_n(t_m) \end{bmatrix} \text{ e } b = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}$$

1.7 Equações Não-Lineares

Sistemas de equações não-lineares aparecem em problemas de otimização, mas também em equações diferenciais e suas formas discretizadas, jogos dinâmicos e processos iterativos. Seja $f(x) : \mathbb{R}^n \to \mathbb{R}^m$ uma função contínua e diferenciável, o problema de interesse é definido como segue:

$$\text{Encontre } x^* \text{ tal que } f(x^*) = 0$$

Alguns algoritmos transformam este problema em um problema de otimização irrestrita:

$$\begin{aligned} \text{Minimize} \quad & \|f(x)\|^2 \\ x \in \mathbb{R}^n & \end{aligned}$$

onde $\| \bullet \|$ é uma norma vetorial.

1.7.1 Aplicação em Sistemas de Controle

Dado um sistema de equações diferenciais, encontre um ponto de equilíbrio. Considere o sistema de equações diferenciais abaixo:

$$\begin{cases} \dot{x}_1 &= -x_1 - 1 \\ \dot{x}_2 &= 2x_1x_2 - 2\sqrt{2}x_1 + 2x_2 - 2\sqrt{2} + sin(x_2 - \sqrt{2}) \\ \dot{x}_3 &= 2x_2 - 2\sqrt{2} \end{cases}$$

De uma forma mais compacta, o sistema acima pode ser escrito como $\dot{x} = f(x)$ onde $x = [x_1, x_2, x_3]$. Um problema típico no domínio de controle não linear é a busca de um ponto de operação ou de equilíbrio, i.e., um ponto x tal que $\dot{x} = 0$. O problema de encontrar um ponto de equilíbrio pode ser reduzido à solução de um sistema de equações não lineares: $f(x) = 0$. Para o exemplo acima, um ponto de equilíbrio é $x^* = [-1, \sqrt{2}, x_3]$.

1.8 Otimização Não-Linear sem Restrições

Otimização irrestrita (sem restrições) constitui um bloco fundamental no desenvolvimento de software. Algoritmos para solução de problemas de otimização restrita fazem uso de otimização irrestrita. O problema de otimização irrestrita é definido como:

$$\begin{array}{c} \text{Minimize} \quad f(x) \\ x \in \mathbb{R}^n \end{array}$$

Tipicamente procura-se um ótimo local x^*, ou seja, x^* tal que $f(x^*) \leqslant f(x)$ para todo $x \in B(x^*, \delta) = \{x : \|x - x^*\| \leqslant \delta\}$ e $\delta > 0$. Otimização global se preocupa em encontrar um vetor x^* cujo valor $f(x^*) \leqslant f(x)$ para todo $x \in \mathbb{R}^n$.

1.8.1 Problema Exemplo

Seja $z = (x, y)$ a coordenada onde será instalada uma central telefônica. Suponha que as chamadas são recebidas de um conjunto $S = \{z_1 = (x_1, y_1), \ldots, z_m = (x_m, y_m)\}$ de localidades com probabilidade uniforme. Seja $\mathbf{Z}$ a variável randômica associada com o local das chamadas. Portanto, Z assume valores do conjunto S tal que a probabilidade de uma chamada vir do local k é $Pr\{Z = z_k\} = 1/m$, para $k = 1, \ldots, m$.

Tarefa

Qual deve ser a localização da central telefônica para que $E[\|Z - z\|^2]$ seja minimizado? Em outras palavras, desejamos minimizar o quadrado da distância da central telefônica aos locais das chamadas. $E[f(Z)]$ é o valor esperado da função $f(Z)$ da variável aleatória Z[2].

[2]Quando a variável aleatória Z assume valores $z_k \in \{z_1, \ldots, z_m\}$, onde a probabilidade de $Z = z_k$ é conhecida e dada por p_k, então o valor esperado de uma função $f(Z)$ é definido por $E[f(Z)] = \sum_{k=1}^{m} f(z_k)p_k$.

Solução

Desenvolvendo a expressão do valor esperado, deduzimos:

$$\begin{aligned} E[\|Z - z\|^2] &= \sum_{k=1}^{m} \|z_k - z\|^2 \times Pr\{Z = z_k\} \\ &= \sum_{k=1}^{m} \|(x_k, y_k) - (x, y)\|^2/m \end{aligned}$$

Assim, o problema pode ser reduzido a um problema de mínimos quadrados, que é um caso particular de otimização irrestrita.

Exemplo Numérico

Para o problema de localizar a central telefônica na região dada pela Figura 1.2 que indica os locais de possíveis chamadas. A chamadas podem ser originadas dos seguintes locais: $z_1 = (-5, 12)$, $z_2 = (-8, 8)$, $z_3 = (-4, 9)$, $z_4 = (-11, 2)$, $z_5 = (-11, -4)$, $z_6 = (-2, -9)$, $z_7 = (5, -4)$, $z_8 = (15, -8)$, $z_9 = (17, 4)$, $z_{10} = (12, 8)$, $z_{11} = (17, 13)$, $z_{12} = (12, 13)$ e $z_{13} = (7, 11)$.

Para a situação onde as chamadas são equiprováveis, a localização ótima da central telefônica é $(3.3846, 4.2308)$.

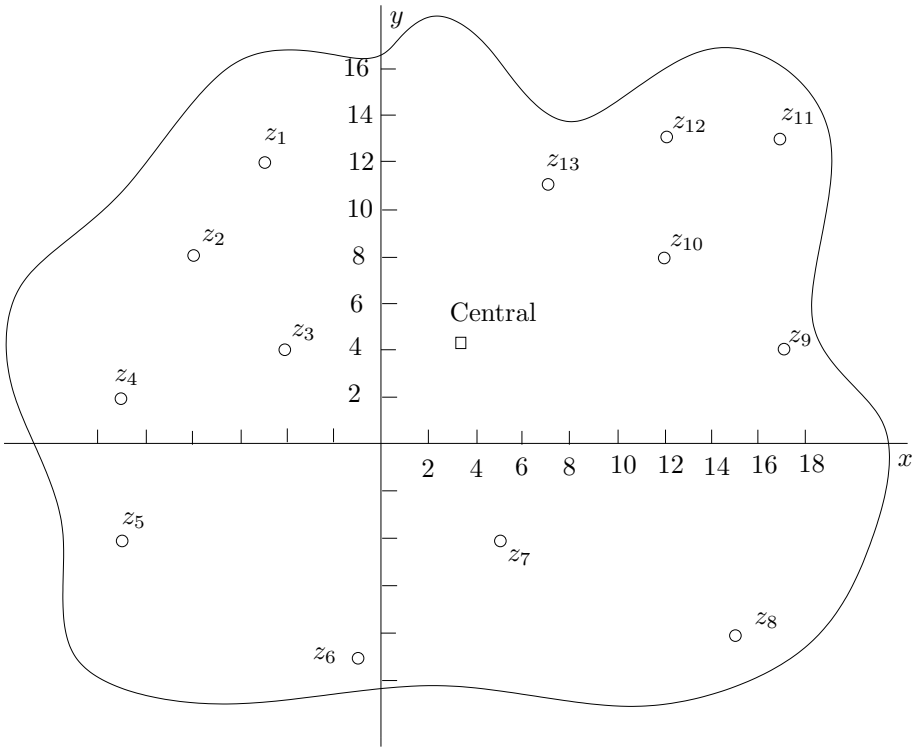


Figura 1.2: Instância do problema de localização de central telefônica.

1.9 Otimização Não-Linear com Limites Superiores/Inferiores

Esta classe de problemas é dada pela formulação a seguir:

$$\begin{array}{ll} \text{Minimize} & f(x) \\ x \in \mathbb{R}^n & \\ \text{Sujeito a :} & l \leqslant x \leqslant u \end{array}$$

onde f é uma função contínua e diferenciável.

Estes modelos têm aplicações em engenharia e na identificação de modelos físicos, onde as grandezas e parâmetros são sujeitos a limites. Alguns algoritmos de otimização restrita resolvem sequências de problemas com limites superiores e inferiores.

1.10 Otimização Não-Linear com Restrições

Os problemas de otimização não-linear com restrições consistem em minimizar uma função não-linear sujeita a restrições não-lineares. O problema geral é da forma:

$$\begin{array}{ll} \text{Minimize} & f(x) \\ \text{Sujeito a :} & \\ & g(x) \leqslant 0 \\ & h(x) = 0 \end{array}$$

onde $f : \mathbb{R}^n \to \mathbb{R}^m$, $g : \mathbb{R}^n \to \mathbb{R}^p$ e $h : \mathbb{R}^n \to \mathbb{R}^q$ são funções contínuas e diferenciáveis. Os modelos de otimização não-linear restritos são os mais gerais no domínio da otimização contínua.

1.10.1 Problema Exemplo

Desejamos instalar uma estação de bombeiros de forma que a mesma esteja dentro de um raio r (km) de um conjunto $S = \{(x_1, y_1), \ldots, (x_p, y_p)\}$ de prédios nas proximidades da unidade dos bombeiros. Além disso, desejamos localizá-la o mais afastado possível de um conjunto $T = \{(\tilde{x}_1, \tilde{y}_1), \ldots, (\tilde{x}_q, \tilde{y}_q)\}$ de estações de bombeiros vizinhas.

Tarefa

Formule este problema em linguagem de otimização.

Formulação

$$\begin{array}{lll} \text{Maximize} & d & \\ \text{Sujeito a :} & & \\ & \|(x, y) - (x_j, y_j)\| \leqslant r & j = 1, \ldots, p \\ & \|(x, y) - (\tilde{x}_j, \tilde{y}_j)\| \geqslant d & j = 1, \ldots, q \end{array}$$

Observações:

$$\begin{aligned} d &\leqslant \min\{\|(x,y)-(\tilde{x}_j,\tilde{y}_j)\| : j=1,\ldots,q\} \\ r &\geqslant \max\{\|(x,y)-(x_j,y_j)\| : j=1,\ldots,p\} \end{aligned}$$

1.11 Programação Semi-Definida

Esta classe compreende problemas com função objetivo linear e restrições envolvendo matrizes e suas propriedades (tais como matriz positiva definida e semi-definida). Tipicamente, uma matriz $F_i(x)$ é definida como uma função afim com parâmetros dados pelo vetor x. Mais precisamente:

$$F_i(x) = F_{i,0} + F_{i,1}x_1 + \ldots + F_{i,n}x_n$$

onde $F_{ij} \in \mathbb{R}^{n\times n}$ é uma matriz simétrica, ou seja, $F_{i,j} = F_{i,j}^T$. Os problemas são expressos na forma a seguir:

$$\begin{array}{ll} \text{Minimize} & c^T x \\ \text{Sujeito a :} & \\ & F_i(x) \succcurlyeq 0 \qquad i=1,\ldots,m \end{array}$$

onde $F_i(x) \succcurlyeq 0$ significa que a matriz $F_i(x)$ deve ser positiva semi-definida (*linear matrix inequality*).

1.11.1 Exemplo

Considere o seguinte sistema dinâmico

$$\dot{x} = Ax$$

Uma condição suficiente para que o sistema convirja para $x^* = 0$ à medida que $t \to \infty$, a partir de qualquer estado inicial $x(0)$, é a existência de uma função $V(x)$ com as seguintes propriedades:

- $V(x)$ é positiva definida, ou seja, para todo $x \neq 0$ tem-se $V(x) > 0$ e $V(0) = 0$
- $\dot{V}(x) < 0$ para $x \neq 0$.

Tal função é conhecida como função Lyapunov.

Como encontrar tal função?

Seja $V(x) = x^T P x$ para $P > 0$. Neste caso $V(x)$ satisfaz a condição de ser positiva definida. Como fazer para satisfazer a segunda condição?

$$\begin{aligned} \frac{d}{dt}V &= \frac{d}{dt}(x^T P x) \\ &= \dot{x}^T P x + x^T P \dot{x} \\ &= (x^T A^T)Px + x^T P(Ax) \\ &= x^T(A^T P + PA)x \end{aligned}$$

Então a condição $\dot{V} < 0$ é equivalente a $A^T P + PA < 0$. O problema de encontrar uma função Lyapunov, quando esta existe, pode ser colocado em programação matemática:

$$\begin{array}{ll} \text{Minimize} & c^T x \\ \text{Sujeito a :} & \\ & P(x) > 0 \\ & A^T P(x) + P(x)A < 0 \end{array}$$

onde $P(x)$ gera o espaço de matrizes simétricas.

1.12 Referências

A modelagem em programação matemática é tanto arte quanto ciência. Pode-se dizer que um problema bem-formulado pode ser resolvido eficientemente com algoritmos de propósito geral. A habilidade de modelar depende da experiência, intuição e conhecimento dos algoritmos que podem ser empregados para resolução. Na prática a busca de um modelo é um processo iterativo: cada modelo é avaliado numericamente e a qualidade da aproximação é medida; o modelo é revisado e o processo se repete. O livro de Williams [43] é recomendado como texto introdutório para modelagem em programação matemática. A Internet também oferece um volume fantástico de referências. Dentre elas, recomendamos:

- Optimization Tree: `http://www-fp.mcs.anl.gov/otc/Guide/OptWeb/`
- OR Notes: `http://www.brunel.ac.uk/depts/ma/research/jeb/or/contents.html`

1.13 Exercícios

EX 1.1 (Produção Industrial Otimizada): Uma companhia manufatura dois produtos (x e y) usando duas máquinas (M_1 e M_2). Para produzir uma unidade do item x, são necessários 50 minutos de processamento na máquina M_1 e 30 minutos de processamento na máquina M_2. Para produzir cada unidade do item y, são necessários 24 minutos de processamento na máquina M_1 e 33 minutos de processamento na máquina M_2. No início da semana, tem-se 30 unidades de x e 90 unidades de y em estoque. O tempo disponível para processamento na máquina M_1 durante a semana é de 40 horas, já para a máquina M_2 o tempo disponível é de 35 horas. A demanda por item x na semana corrente está prevista em 75 unidades, enquanto que para o item y a estimativa é de 95 unidades. O objetivo da empresa é maximizar a quantidade agregada de itens x e y no fim da semana.

Tarefa: formule em programação linear o problema de decidir quantas unidades de cada item devem ser produzidas.

EX 1.2 (Mínimos Quadrados): Formule os problemas a seguir como problemas de mínimos quadrados. Para cada problema forneça a matriz A e o vetor b tal que

o problema possa ser expresso como:

$$\underset{x}{\text{Minimize}} \quad \|Ax - b\|^2$$

Tarefas

a) Minimize $x^T x + \|Bx - d\|^2$, onde $B \in \mathbb{R}^{p \times n}$, e $d \in R^p$ e $x \in R^n$.

b) Minimize $\|Bx - d\|^2 + \|Fx - g\|^2$, onde $B \in \mathbb{R}^{p \times n}$, $d \in \mathbb{R}^p$, $F \in \mathbb{R}^{q \times n}$, $g \in \mathbb{R}^q$ e $x \in R^n$.

c) Minimize $\|Bx - d\|^2 + 2\|Fx - g\|^2$, onde $B \in \mathbb{R}^{p \times n}$, $d \in \mathbb{R}^p$, $F \in \mathbb{R}^{q \times n}$, $g \in \mathbb{R}^q$ e $x \in R^n$.

d) Minimize $x_1^2 + 2x_2^2 + 3x_3^2 + (x_1 - x_2 + x_3 - 1)^2 + (-x_1 - 4x_2 + 2)^2$ (Sugestão: expresse os três termos principais na forma procurada, depois utilize os resultados obtidos em (a), (b) e (c).)

EX 1.3 (Projeto de Latas): O preço do cm^2 de latão é R$ 2,00. Deseja-se produzir latas cilíndricas sem tampa, com volume igual ou superior a 500 cm^3 e de custo mais baixo possível. Formule em programação matemática (otimização não-linear com restrições) o problema de dimensionar a lata, ou seja, de encontrar a altura h (cm) e raio r (cm) da lata. É possível formular este problema em programação não-linear irrestrita?

EX 1.4 (Rede de Abastecimento de Água): Considere uma região onde serão instaladas M caixas de água, sendo a caixa m localizada na posição (x_m, y_m) conhecida. Suponha que um tubo mestre deverá ser instalado horizontalmente, ou seja, ao longo da linha y_t, e que tubos verticais secundários serão construídos para conectar o tubo mestre a cada caixa. Desejamos encontrar a altura para instalar o tubo mestre (valor de y_t) tal que o comprimento total da rede de abastecimento seja o menor possível. Formule este problema em programação matemática. É possível expressar tal problema em programação linear?

EX 1.5 (*The Queens Problem*): Considere um tabuleiro de xadrez (8×8 posições). Deseja-se colocar o maior número possível de rainhas no tabuleiro, de maneira que nenhuma delas possa ser eliminada por nenhuma das demais rainhas. Formule o problema de encontrar tais posições em programação matemática (otimização linear inteira).

EX 1.6 (Rastreamento de Objetos): Dois radares (R_1 e R_2) estão localizados nas posições $z_1 = (x_1, y_1)$ e $z_2 = (x_2, y_2)$ respectivamente. Cada radar informa o centro de rastreamento e controle da respectiva distância a um objeto desconhecido $z = (x, y)$. Sendo d_1 a distância de R_1 a z e d_2 a distância de R_2 a z, formule o problema de calcular as coordenadas $z = (x, y)$ do objeto desconhecido.

(Sugestão: expresse o problema em duas equações não-lineares.)

EX 1.7 (Escalonamento de Tarefas): Um conjunto de n tarefas devem ser processadas em uma máquina capaz de operar uma tarefa de cada vez. Cada tarefa

j consome p_j unidades de tempo na máquina. Um grafo direcionado acíclico $G = (V, E)$, com $V = \{1, \ldots, n\}$, estabelece uma ordem parcial de precedência para execução na máquina, isto é, se existe um caminho $\delta_{i,j}$ de i para j em G, então a tarefa i deve ser processada antes da tarefa j. Dados pesos não negativos $w_j, j = 1, \ldots, n$, em que ordem as tarefas devem ser processadas na máquina de maneira a minimizar a soma dos instantes de início de processamento das tarefas (s_j é o instante em que a tarefa j inicia processamento na máquina) ponderados pelos pesos (w_j), respeitando as restrições de precedência? Formule o problema em programação linear inteira mista (com variáveis discretas e contínuas).

EX 1.8: Considere a formulação do problema de instalação de depósitos, como visto anteriormente:

$$
\begin{array}{lll}
\text{Minimize} & \displaystyle\sum_{i=1}^{n}\sum_{j=1}^{m} c_{ij}x_{ij} + \sum_{j=1}^{m} f_j y_j & \\
\text{Sujeito a :} & & \\
& \displaystyle\sum_{i=1}^{n} d_i x_{ij} \leqslant u_j y_j & j = 1, \ldots, m \\
& \displaystyle\sum_{j=1}^{m} x_{ij} = 1 & i = 1, \ldots, n \\
& x_{ij} \in \mathbb{B} & i = 1, \ldots, n \\
& & j = 1, \ldots, m \\
& y_j \in \mathbb{B} & j = 1, \ldots, m
\end{array}
$$

Existem outras formulações para o problema? Quais são as vantagens e desvantagens das formulações alternativas?

EX 1.9: Para problemas de otimização quadrática cuja matriz Q é simétrica, o Prof. Kunz desenvolveu um algoritmo capaz de encontrar uma solução global ótima. Suponha que você tem que resolver um problema de otimização quadrática onde Q não é simétrica. Será que é possível resolver o problema em questão com o algoritmo do Prof. Kunz?

Capítulo 2

Minimização de Funções com o Método de Descenso

Este capítulo se concentra nas propriedades gerais dos problemas de otimização não-linear sem restrições e no algoritmo de descenso. São introduzidos conceitos de ótimo local e ótimo global, bem como condições necessárias e condições suficientes para que uma solução candidata seja um ótimo local. O algoritmo de descenso e as condições para convergência global a partir de um ponto qualquer para um ótimo local são apresentados na seqüência. Por fim analisa-se a taxa de convergência do algoritmo de descenso.

2.1 Problemas de Interesse

Consideraremos primeiramente o problema de minimização irrestrita:

$$\begin{aligned} P_1: \quad & \text{Minimize } f(x) \\ & x \in \mathbb{R}^n \\ & \text{onde } f : \mathbb{R}^n \to \mathbb{R} \end{aligned}$$

Também investigaremos a solução de equações não-lineares, mas com maior ênfase no próximo capítulo. Este problema pode ser formulado em programação matemática como:

$$\begin{aligned} P_2: \quad & \text{Encontre } x \in \mathbb{R}^n, \text{ tal que } c(x) = 0 \\ & \text{onde } c : \mathbb{R}^n \to \mathbb{R}^m \end{aligned}$$

Vamos estudar conceitos básicos em otimização, apresentando os algoritmos de descenso e de Newton. Os dois algoritmos podem, em princípio, ser utilizados para resolver P_1 e P_2, todavia apresentam propriedades distintas. Assumiremos que as funções f e c são contínuas, diferenciáveis e algumas vezes duas vezes diferenciáveis. Aqui, nos concentramos no algoritmo de descenso para solução de P_1.

2.2 Fundamentos de Otimização Irrestrita

2.2.1 O que é uma solução para P_1?

Em geral, gostaríamos de encontrar uma solução ótima global, ou seja, um ponto x^* tal que $f(x^*) \leqslant f(x)$ para todo $x \in \mathbb{R}^n$. Tipicamente, é extremamente difícil encontrar uma solução global porque o conhecimento de f é usualmente local. A grande maioria dos algoritmos é capaz de encontrar uma solução ótima para uma certa vizinhança de f. Formalmente, dizemos que x^* é um ótimo local se existe uma vizinhança $\mathcal{N}$ de x^* tal que $f(x^*) \leqslant f(x)$ para todo $x \in \mathcal{N}$, onde $\mathcal{N} = \mathcal{N}(x^*, \epsilon) = \{x \in \mathbb{R}^n : \|x - x^*\| \leqslant \epsilon\}$ para algum $\epsilon > 0$ e qualquer norma vetorial $\|\bullet\|$. Um ponto x^* é um ótimo local estrito se $f(x^*) < f(x)$ para todo $x \in \mathcal{N}$, $x \neq x^*$.

Exemplo

Considere a função $f(x) = x^4 \cos(1/x) + 2x^4$ para $x \neq 0$ e $f(0) = 0$. Note que f é contínua e duas vezes diferenciável. O ponto $x^* = 0$ é um ótimo global, mas existe um número infinito de ótimos locais.

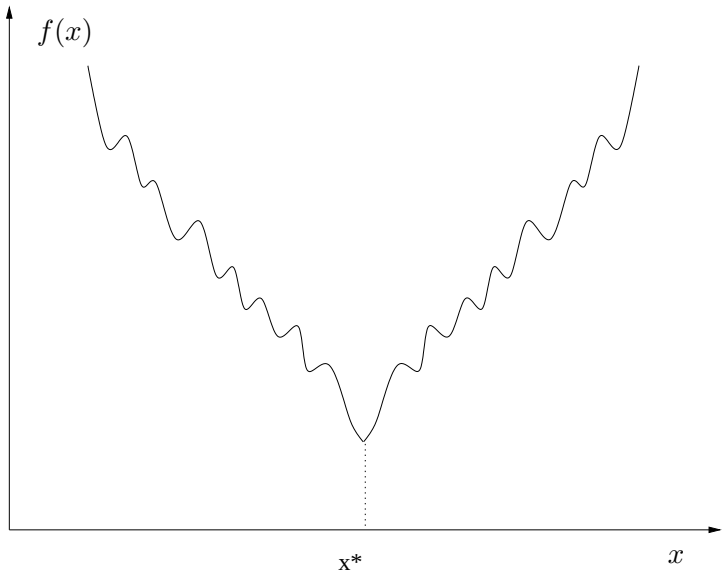


Figura 2.1: Exemplo de função com um número infinito de ótimos locais

2.2.2 Reconhecendo Um Mínimo Local

A partir das definições acima, parece que a única maneira de se identificar um mínimo local é através da comparação com os pontos na vizinhança. Quando a função é suave, entretanto, há formas mais eficientes e práticas de se identificar um mínimo local.

1) Expansão de Taylor de Primeira Ordem

Assumindo que a função f é diferenciável, a expansão de Taylor consiste em aproximar f em torno de um ponto $x \in \mathbb{R}^n$ como:

$$f(x + \Delta x) \approx f(x) + \nabla f(x)^T \Delta x + O(\delta \|\Delta x\|^2).$$

onde $\nabla f(x)^T = [\partial f(x)/\partial x_1, \ldots, \partial f(x)/\partial x_n]$ é o gradiente de f e $O(\delta\|\Delta x\|^2)$ é o resíduo (erro da aproximação). Para pequenos passos ($\|\Delta x\|$ pequeno) podemos desprezar o resíduo, resultando na seguinte aproximação:

$$f(x + \Delta x) \approx f(x) + \nabla f(x)^T \Delta x$$

Teorema 2.1 *(Condição necessária de primeira ordem) Se x^* é um ótimo local e f é uma função continuamente diferenciável em uma vizinhança aberta, então $\nabla f(x^*) = 0$.*

Prova. Vamos supor o contrário. Seja $\Delta x = -\delta \nabla f(x^*)$ para algum $\delta > 0$ pequeno, tal que $x = (x^* + \Delta x)$ esteja na vizinhança de x^*. Portanto, $f(x) = f(x^* + \Delta x) = f(x^*) + \nabla f(x^*)^T[-\delta \nabla f(x^*)] < f(x^*)$. Isto configura uma contradição. □

2) Expansão de Taylor de Segunda Ordem

Se f é duas vezes diferenciável, podemos utilizar uma aproximação de segunda ordem como segue:

$$f(x + \Delta x) \approx f(x) + \nabla f(x)^T \Delta x + \frac{1}{2} \Delta x^T \nabla^2 f(x) \Delta x + O(\delta \|\Delta x\|^3)$$

onde a matriz $\nabla^2 f(x)$ é denominada de Hessiana, sendo sua entrada i, j definida por $\partial^2 f(x)/\partial x_i \partial x_j = \partial[\partial f(x)/\partial x_j]/\partial x_i$. Para um pequeno passo Δx, o resíduo pode ser desprezado resultando na expansão abaixo:

$$f(x + \Delta x) \approx f(x) + \nabla f(x)^T \Delta x + \frac{1}{2} \Delta x^T \nabla^2 f(x) \Delta x$$

Teorema 2.2 *(Condição necessária de segunda ordem) Se x^* é um ótimo local de f e $\nabla^2 f$ é contínua em uma vizinhança aberta de x^*, então $\nabla f(x^*) = 0$ e $\nabla^2 f(x^*)$ é positiva semi-definida.*

Prova. Pelo Teorema 2.1, $\nabla f(x^*) = 0$. Pela aproximação de Taylor de segunda ordem, podemos dizer que $f(x^* + \Delta x) = f(x^*) + \frac{1}{2} \Delta x^T \nabla^2 f(x^*) \Delta x$. Suponha que $\nabla^2 f(x^*)$ não é positiva semi-definida, o que implica que existe Δx pequeno tal que $\Delta x^T \nabla^2 f(x^*) \Delta x < 0$. Portanto, $f(x^* + \Delta x) = f(x^*) + \frac{1}{2} \Delta x^T \nabla^2 f(x^*) \Delta x < f(x^*)$ e, conseqüentemente, x^* não pode ser um ótimo local de f. □

Teorema 2.3 *(Condição suficiente de segunda ordem) Suponha que $\nabla^2 f(x)$ seja contínua em uma vizinhança de x^*, $\nabla f(x^*) = 0$ e $\nabla^2 f(x^*)$ seja positiva definida. Então x^* é um ótimo local estrito de f.*

Prova. Seja $x = x^* + \Delta x$ um ponto na vizinhança de x^*. Para Δx suficientemente pequeno, $f(x) = f(x^* + \Delta x) = f(x^*) + \nabla f(x^*)^T \Delta x + \frac{1}{2}\Delta x^T \nabla^2 f(x^*)\Delta x = f(x^*) + \frac{1}{2}\Delta x^T \nabla^2 f(x^*)\Delta x > f(x^*)$, porque $\nabla f(x^*) = 0$ e $\nabla^2 f(x^*)$ é positiva definida. A última desigualdade implica que todo x na vizinhança de x^*, $x \in \mathbb{R}^n$ tal que $\|x - x^*\| \leqslant \delta$ para $\delta > 0$ pequeno o suficiente, induz $f(x) > f(x^*)$. □

Observação: as condições suficientes de segunda ordem não são necessárias, ou seja, uma solução pode ser um mínimo estrito e mesmo assim não passar no teste de suficiência. Um exemplo simples é a função $f(x) = x^4$. Para esta função, o ponto $x^* = 0$ é um mínimo local estrito cuja Hessiana não é positiva definida.

3) Funções Convexas

Uma função $f : \mathbb{R}^n \to \mathbb{R}$ é convexa se para todo $x, y \in \mathbb{R}^n$ e $\alpha \in (0, 1)$ tem-se:

$$f(\alpha x + (1 - \alpha)y) \leqslant \alpha f(x) + (1 - \alpha) f(y)$$

Se f é uma função duas vezes diferenciável, então f é convexa se $\nabla^2 f(x)$ é positiva semi-definita para todo $x \in \mathbb{R}^n$. Uma função f é côncava se $-f$ é convexa.

Teorema 2.4 *Se f é convexa, então todo o mínimo local é também um mínimo global.*

Prova. Seja x^* um mínimo local e suponha, por absurdo, que x^* não seja um mínimo global. Seja $\tilde{x}$ um mínimo global e considere o segmento de reta que une os pontos x^* e $\tilde{x}$, ou seja,

$$x = \alpha \tilde{x} + (1 - \alpha)x^* \text{ onde } \alpha \in (0, 1].$$

Por convexidade de f, temos que $f(x) \leqslant \alpha f(\tilde{x}) + (1 - \alpha) f(x^*) < \alpha f(x^*) + (1 - \alpha) f(x^*) = f(x^*)$. Qualquer vizinhança $\mathcal{N}$ de x^* contém um pedaço do segmento de reta acima definido e, portanto, sempre existirá um x em $\mathcal{N}$ que satisfaça a desigualdade $f(x) < f(x^*)$. Conclui-se que x^* não é um ótimo local. □

2.3 O Algoritmo de Descenso

A grande maioria dos algoritmos de otimização são iterativos. Dado um ponto x_0, eles produzem uma seqüência de pontos $x_0, x_1, x_2, \ldots, x_T = \{x_k\}$ convergente para um mínimo local. Tipicamente, um processo iterativo consiste em obter uma nova solução a partir da solução anterior:

$$x_{k+1} = G_k(x_k) \qquad \text{para } k = 0, \ldots, T - 1$$

Para que a seqüência convirja para um ótimo local, os algoritmos induzem uma redução (não necessariamente a cada iteração) da função objetivo, ou seja, $f(x_k) < f(x_{k-m})$.

2.3.1 O Algoritmo de Descenso em Detalhes

O algoritmo de descenso consiste em escolher uma direção p_k, a cada iteração k, e fazer uma busca ao longo desta direção a partir do iterando x_k. Em outras palavras, o algoritmo resolve o seguinte problema de uma dimensão:

$$\begin{aligned} &\text{Minimize} \quad \phi(\alpha_k) = f(x_k + \alpha_k p_k) \\ &\alpha_k > 0 \end{aligned} \tag{2.1}$$

onde α_k é o passo a ser dado na direção p_k de forma que o próximo iterando passa a ser $x_{k+1} = x_k + \alpha_k p_k$.

A solução exata de (2.1) induziria o benefício máximo da direção p_k; todavia, a minimização exata é cara computacionalmente e não necessária. Em vez disso, o método de descenso gera um número limitado de passos até encontrar um passo α_k que aproxima o mínimo de (2.1). Um novo ponto é gerado, $x_{k+1} = x_k + \alpha_k p_k$, uma nova direção p_{k+1} é encontrada e o processo se repete.

2.3.2 Direção de Busca

A direção de maior descenso, $-\nabla f(x_k)$, é a opção óbvia para a busca direcional. A direção de maior descenso produz a maior taxa de redução no valor de f dentre todas as possíveis direções. O método de maior descenso (*steepest descent*) executa uma busca ao longo da direção $p_k = -\nabla f(x_k)$ a cada iteração. O passo α_k pode ser escolhido de várias formas. Uma vantagem do método de descenso é que ele requer apenas o gradiente, $\nabla f(x_k)$, não necessitando da Hessiana. Todavia, ele pode se tornar excessivamente lento em problemas difíceis.

A Figura 2.2 ilustra as curvas de nível (*contour curves*) de uma função $f : \mathbb{R}^2 \to \mathbb{R}$ em torno de um ótimo local x^*. O gradiente de f em um ponto x é ortogonal à curva de nível que passa pelo ponto x.

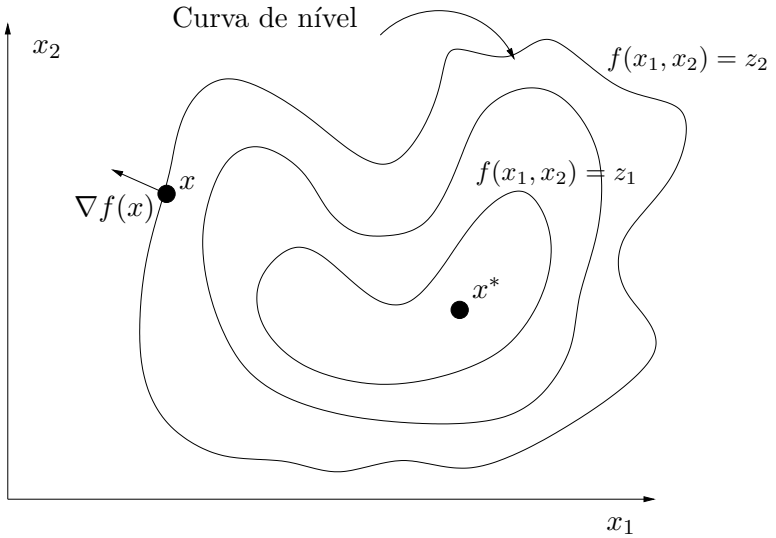


Figura 2.2: Ilustração do gradiente de uma função, $\nabla f(x_k)$, que assume a direção ortogonal à curva de nível que passa por x_k.

2.3.3 Encontrando a Direção de Busca

O processo iterativo do método de descenso é dado por: $x_{k+1} = x_k + \alpha_k p_k$, onde α_k é um escalar positivo (chamado de passo) e p_k é um vetor (chamado de direção de descenso). O sucesso do método depende da escolha adequada da direção e do passo. O algoritmo exige que p_k seja uma direção de descenso, ou seja, $\nabla f(x_k)^T p_k < 0$. Outros algoritmos utilizam uma direção da forma $p_k = -B_k^{-1}\nabla f(x_k)$, onde B_k é uma matriz simétrica não-singular. Note que quando B_k é positiva definida, p_k define uma direção de descenso.

2.3.4 Encontrando o Passo

Durante a computação do passo α_k, deparamo-nos com um conflito. Por um lado, desejamos escolher α_k que produza uma redução substancial em f mas, por outro lado, não desejamos gastar muito esforço computacional durante a escolha. O passo ideal minimizaria a função $\phi(\alpha)$, definida por: $\phi(\alpha) = f(x_k + \alpha p_k)$, onde $\alpha > 0$. Encontrar um minimizador local pode ser muito caro, necessitando várias computações de f e ∇f. Métodos práticos executam uma busca inexata, procurando obter uma redução satisfatória de f a um custo mínimo. Tipicamente, os métodos aproximados estabelecem condições para um decréscimo satisfatório.

Condições para Passo Satisfatório

A condição de Armijo é definida pela seguinte desigualdade:

$$f(x_k + \alpha_k p_k) \leqslant f(x_k) + c_1 \alpha_k \nabla f(x_k)^T p_k \tag{2.2}$$

onde $c_1 \in (0, 1)$. Em outras palavras, a condição estabelece que a redução em f deve ser proporcional ao passo α_k e à derivada direcional $\nabla f(x_k)^T p_k$. Seja $l(\alpha) = f(x_k) + c_1\alpha\nabla f(x_k)^T p_k$, então a condição de Armijo pode ser expressa como $f(x_k + \alpha p_k) \leqslant l(\alpha)$. Note que a função $l(\alpha)$ é uma função decrescente e linear em α. A Figura 2.3 ilustra a condição de Armijo (2.2).

A condição de Armijo não é, entretanto, suficiente para garantir progresso razoável e satisfatório do algoritmo, pois a condição pode ser satisfeita por passos extremamente pequenos. Para evitar passos muito pequenos, uma condição de curvatura é introduzida, a qual exige que:

$$\nabla f(x_k + \alpha_k p_k)^T p_k \geqslant c_2 \nabla f(x_k)^T p_k, \tag{2.3}$$

para alguma constante $c_2 \in (c_1, 1)$. Note que o lado esquerdo é simplesmente $d\phi(\alpha)/d\alpha = \phi'(\alpha)$[1], assim a condição de curvatura garante que a taxa de decrescimento $\phi'(\alpha)$ seja c_2 vezes o gradiente $\phi'(0)$. Em outras palavras, $\phi'(\alpha)$ deve ser menos negativo do que $\phi'(0)$. Isso faz sentido: se $\phi'(0)$ é bastante negativo, então temos uma indicação de que podemos reduzir f substancialmente ao longo da direção escolhida; caso contrário, se $\phi'(0)$ é pouco negativo, temos um sinal de que não podemos reduzir f substancialmente. Passos satisfatórios que satisfazem simultaneamente as condições de Armijo (2.2) e de curvatura (2.3) são ilustrados na Figura 2.4

[1]Use a regra da cadeia para verificar esta afirmação: $dz(y(x))/dx = (dz/dy)(dy/dx)$, onde $z(y)$ é uma função de y e $y(x)$ é uma função de x.

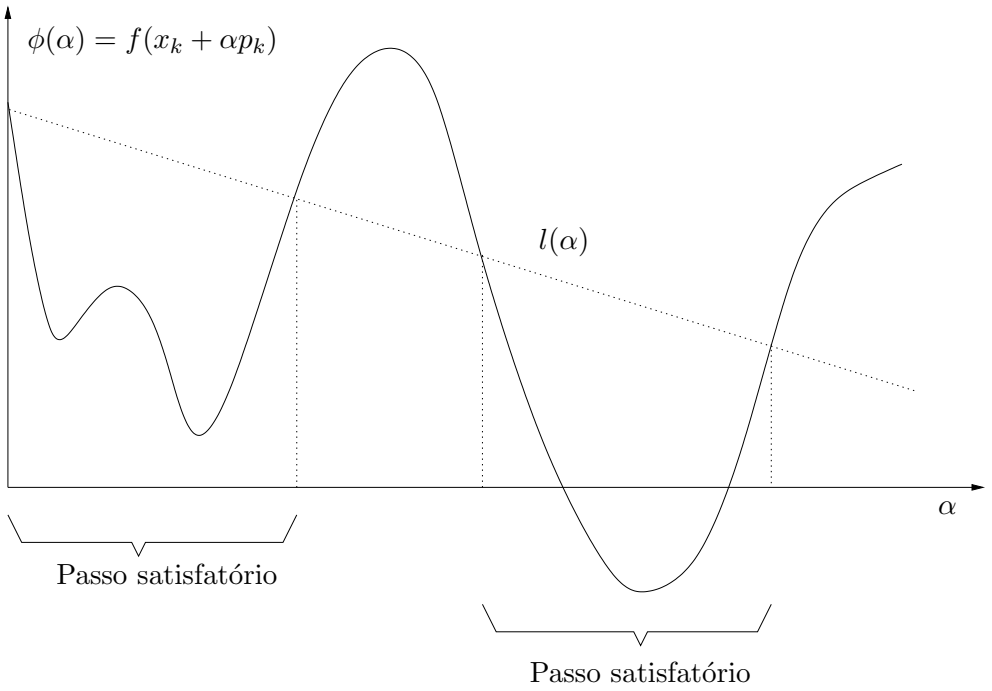


Figura 2.3: Ilustração de passo satisfatório.

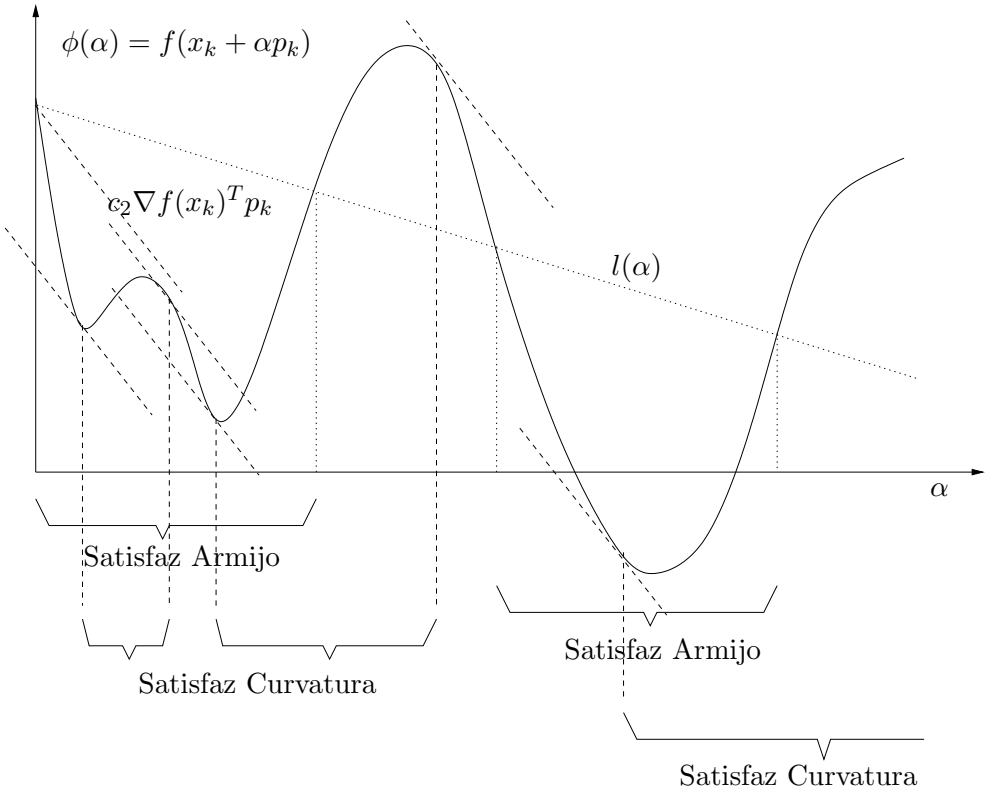


Figura 2.4: Ilustração das condições de Armijo e de curvatura.

As condições suficientes para redução da função objetivo (2.2) e de curvatura (2.3) são conhecidas como condições de Wolfe:

$$\begin{cases} f(x_k + \alpha_k p_k) & \leqslant & f(x_k) + c_1 \alpha_k \nabla f(x_k)^T p_k \\ \nabla f(x_k + \alpha_k p_k)^T p_k & \geqslant & c_2 \nabla f(x_k)^T p_k \end{cases} \tag{2.4}$$

onde $0 < c_1 < c_2 < 1$. Equivalentemente, as condições de Wolfe podem ser expressas como:

$$\begin{cases} \phi(\alpha_k) & \leqslant & \phi(0) + c_1 \alpha_k \phi'(0) \\ \phi'(\alpha_k) & \geqslant & c_2 \phi'(0) \end{cases} \tag{2.5}$$

Não é difícil provar que existem passos α_k que satisfazem as condições de Wolfe se a função f é suave e possui um limite inferior.

Lema 2.1 *Seja $f : \mathbb{R}^n \to \mathbb{R}$ uma função continuamente diferenciável. Seja p_k uma direção de descenso para o ponto x_k, e assuma que f é limitada inferiormente ao longo do raio $\{x_k + \alpha p_k : \alpha > 0\}$. Então, se $0 < c_1 < c_2 < 1$, existem intervalos para os passos que satisfazem as condições de Wolfe.*

Prova. Uma vez que $\phi(\alpha) = f(x_k + \alpha p_k)$ é limitada inferiormente para todo $\alpha > 0$ e uma vez que $0 < c_1 < c_2 < 1$, a linha $l(\alpha) = f(x_k) + \alpha c_1 \nabla f(x_k)^T p_k$ deve interceptar a curva $\phi(\alpha)$ pelo menos uma vez. Seja $\alpha' > 0$ o menor valor α tal que $l(\alpha)$ intercepta $\phi(\alpha)$, ou seja,

$$f(x_k + \alpha' p_k) \quad = \quad f(x_k) + \alpha' c_1 \nabla f_k^T p_k \tag{2.6}$$

A condição de Armijo é claramente satisfeita por todos os passos menores que α'.

A partir do Teorema do Valor Médio[2], deve existir $\alpha'' \in (0, \alpha')$ tal que[3]

$$f(x_k + \alpha' p_k) - f(x_k) = \alpha' \nabla f(x_k + \alpha'' p_k)^T p_k \tag{2.7}$$

Combinando (2.6) e (2.7), obtemos

$$\nabla f(x_k + \alpha'' p_k)^T p_k = c_1 \nabla f_k^T p_k > c_2 \nabla f_k^T p_k \tag{2.8}$$

uma vez que $c_1 < c_2$ e $\nabla f_k^T p_k < 0$. Assim, α'' satisfaz as condições de Wolfe, e as desigualdades (2.4) são estritas. Portanto, de acordo com a propriedade de f ser suave, deve existir um intervalo em torno de α'' para o qual as condições de Wolfe são satisfeitas. □

2.3.5 Redução Suficiente e Retrocesso

Utilizando-se o chamado método de retrocesso, podemos sistematicamente encontrar um passo α_k que satisfaz as condições de Wolfe (2.4) sem considerar a segunda condição.

[2]O teorema do valor médio diz que $f(y) - f(x) = \nabla f(z)^T (y - x)$ sendo z um vetor que pertence ao segmento $x - y$.

[3]Pelo Teorema do Valor Médio, faça $y = x_k + \alpha' p_k$, $x = x_k$ e observe que $y - x = \alpha' p_k$ e $z = x_k + \alpha'' p_k$.

Procedimento

Escolha $\alpha' > 0$ (tipicamente $\alpha' = 1$)
Escolha ρ, $c_1 \in (0, 1)$
Faça $\alpha \leftarrow \alpha'$
Repita até que $f(x_k + \alpha p_k) \leqslant f(x_k) + c_1 \alpha \nabla f(x_k)^T p_k$
    Faça $\alpha \leftarrow \rho\alpha$
Fim-repita
Termine com $\alpha_k \leftarrow \alpha$

O procedimento acima encontra um passo satisfatório (i.e., que satisfaz as condições de Wolfe) após um número finito de iterações pois α_k eventualmente se torna pequeno o suficiente para satisfazer a condição de Armijo.

2.3.6 Convergência do Método de Descenso

Para que o algoritmo de descenso obtenha convergência global[4] devemos escolher não apenas direções apropriadas, mas também passos adequados. O teorema a seguir descreve condições suficientes para que o método de descenso convirja, a partir de qualquer ponto inicial, para um mínimo local.

Teorema 2.5 *Considere uma iteração (2.1), onde p_k é uma direção de descenso e α_k satisfaz as condições de Wolfe (2.4). Suponha que f é limitada por baixo em $\mathbb{R}^n$ e que f é continuamente diferenciável em um conjunto aberto $\mathcal{N}$ que contém o conjunto $\mathcal{L} = \{x : f(x) \leqslant f(x_0)\}$, onde x_0 é o ponto inicial da iteração. Assuma ainda que ∇f é Lipschitz contínua em $\mathcal{N}$, i.e., existe uma constante $L > 0$ tal que*

$$\|\nabla f(x) - \nabla f(\tilde{x})\| \quad \leqslant \quad L\|x - \tilde{x}\|, \text{ para todo } x, \tilde{x} \in \mathcal{N}. \tag{2.9}$$

Então

$$\sum_{k \geqslant 0} \cos(\theta_k)^2 \|\nabla f_k\|^2 \quad < \quad \infty. \tag{2.10}$$

Prova. A partir da iteração $x_{k+1} = x_k + \alpha_k p_k$ e a segunda condição de Wolfe (2.4), $\nabla f(x_k + \alpha_k p_k)^T p_k \geqslant c_2 \nabla f(x_k)^T p_k$, temos que

$$(\nabla f_{k+1} - \nabla f_k)^T p_k \geqslant (c_2 - 1)\nabla f_k^T p_k$$

enquanto que a condição Lipschitz e a desigualdade de Schwartz[5] implicam

$$\begin{aligned}
(\nabla f_{k+1} - \nabla f_k)^T p_k &\leqslant |(\nabla f_{k+1} - \nabla f_k)^T p_k| \\
&\leqslant \|\nabla f_{k+1} - \nabla f_k\| \|p_k\| \\
&\leqslant L\|x_k + \alpha_k p_k - x_k\| \|p_k\| \\
&\leqslant \alpha_k L \|p_k\|^2.
\end{aligned}$$

[4]Por convergência global entendemos a propriedade do algoritmo de convergir a partir de qualquer ponto inicial x_0 para um ótimo local x^*

[5]Para $\| \bullet \| = \| \bullet \|_2$, $|x^T y| \leqslant \|x\| \|y\|$

Combinando as duas relações acima, obtemos:

$$\alpha_k \geqslant \frac{c_2 - 1}{L} \frac{\nabla f_k^T p_k}{\|p_k\|^2}.$$

Substituindo esta desigualdade na primeira condição de Wolfe, verificamos

$$f_{k+1} \leqslant f_k - c_1 \frac{1 - c_2}{L} \frac{(\nabla f_k^T p_k)^2}{\|p_k\|^2}.$$

Fazendo θ_k ser o ângulo entre p_k e a direção de maior descenso $-\nabla f_k$, verifica-se que

$$\cos \theta_k = \frac{-\nabla f_k^T p_k}{\|\nabla f_k\| \|p_k\|}.$$

Fazendo $c = c_1(1 - c_2)/L$ e combinando as duas últimas expressões obtemos

$$f_{k+1} \leqslant f_k - c \cos^2 \theta_k \|\nabla f_k\|^2.$$

Somando a expressão acima para todos os índices menores ou iguais a k, obtemos

$$f_{k+1} \leqslant f_0 - c \sum_{j=0}^{k} \cos^2 \theta_j \|\nabla f_j\|^2.$$

Uma vez que f é limitada inferiormente, sabemos que $f_0 - f_{k+1}$ é uma constante positiva, para todo k. Tomando o limite da expressão acima, concluímos que

$$\sum_{j=0}^{\infty} \cos^2 \theta_j \|\nabla f_j\|^2 < \infty.$$

□

A conseqüência do Teorema 2.5 é a convergência global do método de descenso, visto que este sempre escolhe como direção de descenso um vetor p_k com ângulo θ_k que se afasta de 90^0. Assim $\cos \theta_k \geqslant \delta > 0$ para todo k. Conseqüentemente $\lim_{k \to \infty} \|\nabla f_k\| = 0$. Em outras palavras, a norma do gradiente $\|\nabla f_k\|$ converge para zero se a direção de descenso não é muito próxima da ortogonalidade e as condições de Wolfe são satisfeitas.

2.3.7 Taxa de Convergência do Método de Descenso Íngreme

Aqui vamos considerar a taxa de convergência de métodos de descenso em regiões próximas de um ótimo local. Uma função continuamente diferenciável f pode ser aproximada em torno de um ótimo local x^* não singular[6] como uma função quadrática

$$f(x) = f(x^*) + \frac{1}{2}(x - x^*)^T \nabla^2 f(x^*)(x - x^*)$$

[6] x^* é dito ótimo local não singular se as condições suficientes de otimalidade local de segunda ordem são satisfeitas.

desde que $\|x - x^*\|$ seja pequeno. Podemos assumir sem perda de generalidade que $x^* = 0$ e $f(x^*) = 0$[7] . Assim a aproximação quadrática de f fica

$$f(x) = \frac{1}{2}x^T Qx, \quad \nabla f(x) = Qx, \quad \nabla^2 f(x) = Q.$$

O método de descenso íngreme ou de descenso máximo (*steepest descent*) pode ser colocado na forma:

$$x_{k+1} = x_k - \alpha_k \nabla f(x_k) = (I - \alpha_k Q)x_k$$

o que nos leva a

$$\|x_{k+1}\|^2 = x_k^T (I - \alpha^k Q)^2 x_k.$$

Uma vez que $x^T(I - \alpha_k Q)^2 x \leqslant Max\{\lambda \in \sigma((I - \alpha_k Q)^2)\}\|x\|^2$, onde $\sigma(A)$ denota o conjunto de autovalores da matriz A, concluímos que:

$$\|x_{k+1}\|^2 \leqslant Max\{\lambda \in \sigma((I - \alpha_k Q)^2)\}\|x_k\|^2.$$

Os autovalores de $(I - \alpha_k Q)^2$ são iguais a $(1 - \alpha_k \lambda_i)^2$, onde λ_i são os autovalores de Q. Dessa forma podemos deduzir que

$$Max\{\lambda \in \sigma((I - \alpha_k Q)^2)\} = Max\{(1 - \alpha_k m)^2, (1 - \alpha_k M)^2\}$$

onde m (M) é o menor (maior) autovalor de Q. Segue então que para $x_k \neq 0$

$$\frac{\|x_{k+1}\|}{\|x_k\|} \quad \leqslant \quad Max\{|1 - \alpha_k m|, |1 - \alpha_k M|\}. \tag{2.11}$$

Se $|1 - \alpha_k m| < |1 - \alpha_k M|$ então a desigualdade (2.11) é satisfeita como uma equação e x_k é proporcional ao autovetor correspondente a m. Se $|1 - \alpha_k m| > |1 - \alpha_k M|$ então a desigualdade (2.11) é satisfeita como uma equação e x_k é proporcional ao autovetor correspondente a M. O passo α_k que minimiza o limite dado em (2.11) é

$$\alpha^* = \frac{2}{M + m}$$

o que implica em

$$\frac{\|x_{k+1}\|}{\|x_k\|} \leqslant \frac{M - m}{M + m}, \tag{2.12}$$

que é a melhor taxa de convergência do *steepest descent* com passo constante. De acordo com (2.12), quanto menor for a diferença entre o maior e menor autovalor de Q, mais rápida é a convergência. Nas situações em que as curvas de nível em torno de x^* são alongadas, i.e., quando a diferença entre o maior e menor autovalor é grande, a taxa de convergência tende a diminuir.

[7]Basta fazer uma mudança de variável $\bar{x} = x - x^*$ deslizando o ponto x^* para a origem do sistema de coordenadas resultante; e adicionar a constante $-f(x^*)$ à função objetivo.

2.4 Referências

Este capítulo foi produzido a partir dos textos de Bertsekas [9] e Nocedal e Wright [39]. O primeiro texto é bastante completo e trata da parte técnica em profundidade, além de apresentar várias aplicações e problemas com inspiração física. O segundo texto é mais introdutório e recomendado para o leitor iniciante que deseja se aprofundar em algoritmos de otimização não-linear. Outras referências com boa cobertura da teoria e algoritmos de otimização não-linear são os livros de Gill et al. [22] e Bazaraa et al. [8].

2.5 Exercícios

EX 2.1: Calcule o gradiente $\nabla f(x)$ e a matriz Hessiana $\nabla^2 f(x)$ da função $f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$. Mostre que o ponto $x^* = (1, 1)$ é o único ótimo global. O ponto x^* satisfaz as condições necessárias para ser um ótimo local?

EX 2.2 Mostre que a função $f(x) = 8x_1 + 12x_2 + x_1^2 - 2x_2^2$ possui apenas um ponto estacionário e que este não define um mínimo nem um máximo de f.

EX 2.3: Seja $c \in \mathbb{R}^n$ um vetor e $A \in \mathbb{R}^{n\times n}$ uma matriz simétrica. Calcule o gradiente e a matriz Hessiana de $f_1(x) = c^T x$ e de $f_2(x) = x^T Ax/2$.

EX 2.4: Define-se uma função $f(x)$ como convexa se, e somente se, $f(\alpha x + (1-\alpha)y) \leqslant \alpha f(x) + (1-\alpha)f(y)$ para todo x, y e $\alpha \in [0, 1]$. Sejam $f(x)$ e $g(x)$ duas funções convexas. Responda às seguintes questões:

a) $h(x) = f(x) + g(x)$ é convexa?

b) $h(x) = f(x) - g(x)$ é convexa?

c) $h(x) = f(x)^2$ é convexa?

d) $h(x) = \sqrt{f(x)}$ é convexa, assumindo que $f(x) \geqslant 0$?

EX 2.5: Seja $f(x)$ uma função quadrática: $f(x) = \frac{1}{2}x^T Qx - b^T x$, onde Q é simétrica e positiva definida. O algoritmo de descenso pode ser descrito como segue:

Seja $x_0 \in \mathbb{R}^n$ o ponto inicial, $k = 0$;
Enquanto $(\|\nabla f(x_k)\| > \sigma)$ e $(k < itbound)$ faça
  $p_k = -\nabla f(x_k)$
  $x_{k+1} = x_k + \alpha_k p_k$, onde $\alpha_k = argmin_\alpha\{f(x_k + \alpha p_k) : \alpha \geqslant 0\}$
  $k = k + 1$
Fim

Implemente esse algoritmo em Matlab. Sua função script deve receber como parâmetros de entrada Q, b, x_0 e gerar a seqüência $\{x_k\}$. Use como tolerância $\sigma = 10^{-6}$ e $itbound = 100$. Rode experimentos para exemplos onde $n = 2$ e $b = 0$, com as seguintes matrizes:

a) uma matriz Q cujo *condition number* $\kappa(Q) = 10$, e

b) uma matriz Q para a qual $\kappa(Q) = 100$.

Para construir a matriz Q tome $D = [0.1\ 0;\ 0\ 1]$, escolha uma matriz randômica e ortogonal P ($P^TP = I$), e finalmente faça $Q = P^TDP$ e $x_0 = P[1;\ 0.1]$; para a parte (b) faça o mesmo, mas utilize 0.01 em vez de 0.1 nos dois lugares.

Apresente os resultados em tabelas bem como em figuras, juntamente com os dados utilizados nos experimentos (Q, x_0, etc.).

EX 2.6: Para $\phi(\alpha) = f(x_k+\alpha p_k)$, mostre que $\phi'(\alpha) = \nabla f(x_k+\alpha p_k)^T p_k$ sendo $Q = Q^T$.

EX 2.7: Mostre que $x^T(I - \alpha_k Q)^2 x \leqslant \max\{\lambda \in \sigma((I - \alpha_k Q)^2)\}\|x\|^2$ sendo $Q = Q^T$.

EX 2.8: Mostre que os autovalores de $(I - \alpha_k Q)^2$ são iguais a $(1 - \alpha_k\lambda_i)^2$, onde λ_i são os autovalores de Q.

EX 2.9: Para $M \geqslant m > 0$, encontre a solução ótima do problema:

$$\begin{array}{ll} \min & \max\{|1 - \alpha m|, |1 - \alpha M|\} \\ \alpha \geqslant 0 & \end{array}$$

EX 2.10: Considere a seguinte forma do algoritmo de descenso. Dado um vetor com solução candidata x_k, um passo inicial s e a direção de descenso $d_k = -\nabla f(x_k)$, se a solução $x_{k+1} = x_k + sd_k$ não produz um decréscimo suficiente de f, então o passo é reduzido até que o decréscimo seja suficiente.

Dados escalares s, β e $\sigma > 0$ com $\beta \in (0, 1)$ e $\sigma \in (0, 1)$, definimos $\alpha_k = \beta^{m_k} s$ onde m_k é o primeiro inteiro não negativo para o qual:

$$f(x_k) - f(x_k + \beta^m s d_k) \geqslant -\sigma\beta^m s \nabla f(x_k)^T d_k, \tag{2.13}$$

obtendo como próximo iterando o ponto $x_{k+1} = x_k + \alpha_k d_k$. Em outras palavras, os passos $\beta^m s$, $m = 0, 1, \ldots$ são realizados até que a desigualdade acima seja satisfeita para $m = m_k$. Portanto, não estamos satisfeitos apenas com uma melhora no valor da função objetivo; a redução deve ser suficientemente grande conforme a desigualdade (2.13).

Considere o problema de minimizar $f(x, y) = 3x^2 + y^4$. As tarefas a serem realizadas são:

a) Aplique uma iteração do algoritmo de descenso acima delineado com o ponto inicial $x_0 = (1, -2)$, $s = 1$, $\sigma = 0.1$ e $\beta = 0.5$. Indique os valores de $x_0 + \beta^m s d_0$, $f(x_0 + \beta^m s d_0)$, $f(x_0) - f(x_0 + \beta^m s d_0)$ e $-\sigma\beta^m s \nabla f(x_0)^T d_0$.

b) Repita os passos de (a) usando $s = 1$, $\sigma = 0.1$ e $\beta = 0.1$. Como que o custo do novo iterando compara com aquele obtido em (a)? Comente sobre os benefícios e perdas obtidos através da escolha de β.

EX 2.11: São dadas duas tarefas:

a) Encontre o paralelepípido de volume unitário que tenha a menor área. Defina x, y e z como as dimensões do paralelepípido.

b) Dado um triângulo no plano, considere o problema de encontrar o ponto cuja soma das distâncias dos vértices do triângulo é mínima. Mostre que tal ponto é um vértice do triângulo ou o ponto que forma um ângulo de 120^o com cada lado do triângulo.

EX 2.12: Responda se as afirmações abaixo são corretas.

i. Prof. Kunz afirma que não existe mínimo local para a função $f(x) = \frac{e^x}{e^x+e^{-x}}$.

ii. Seja $f : \mathbb{R}^n \to \mathbb{R}$ uma função contínua e diferenciável. Prof. Kunz afirma que se $x \in \mathbb{R}^n$ é um mínimo local, então obrigatoriamente $\nabla f(x) = 0$.

iii. Seja $f : \mathbb{R}^n \to \mathbb{R}$ uma função contínua e diferenciável. Prof. Kunz afirma que se existe um mínimo local, então deve existir um mínimo global.

iv. Seja $f : \mathbb{R} \to \mathbb{R}$ uma função contínua e duas vezes diferenciável. Para um certo $x \in \mathbb{R}$, tem-se $f'(x) = 0$ e $f''(x) \geqslant 0$. Prof. Kunz afirma que x é um mínimo local.

v. Seja $f : \mathbb{R} \to \mathbb{R}$ uma função contínua e duas vezes diferenciável tal que $f''(x) \geqslant 0$ para todo $x \in \mathbb{R}$. Prof. Kunz afirma que se x é um mínimo local, então obrigatoriamente x também é um ótimo global.

vi. Considere a função $f(x) = e^{-x}/x^2$. Dejamos encontrar o mínimo global de $f(x)$ na região $R = [1, \infty)$. Prof. Kunz diz que se $x \in R$ é um mínimo local para $f(x)$ na região R, então x é obrigatoriamente um mínimo global.

vii. Sejam $f(x, y)$ e $g(x, y)$ duas funções contínuas e diferenciáveis. Prof. Kunz afirma que se $(\hat{x}, \hat{y})$ é um mínimo local de $h(x, y) = f(x, y) + g(x, y)$ então $(\hat{x}, \hat{y})$ é também um mínimo local para f e g.

Capítulo 3

Método de Newton: Solução de Equações Não-Lineares e Minimização de Funções

Este capítulo trata do método de Newton e suas aplicações na solução de sistemas de equações não-lineares e na minimização de funções. São apresentados os fundamentos do algoritmo, exemplos e a prova de convergência local em uma variável. Para garantir convergência global do método quando aplicado à minimização de funções, aplica-se a técnica de região de confiança que é expandida e reduzida de forma que a aproximação quadrática da função seja satisfatória dentro da região. A seção final do capítulo estende o método de Newton para o problema de minimização de funções sob restrições de igualdade. Esta seção pode ser omitida sem perda, pois a otimização de funções sob restrições será tópico de outros capítulos.

3.1 Problemas de Interesse

Desenvolveremos métodos de Newton para a solução de equações não-lineares, formuladas como segue:

$$\begin{array}{ll} P_2: & \text{Encontre } x \in \mathbb{R}^n, \text{ tal que } c(x) = 0 \\ & \text{onde } c : \mathbb{R}^n \to \mathbb{R}^m \end{array}$$

Também investigaremos a solução do problema de minimização irrestrita, ou seja:

$$\begin{array}{ll} P_1: & \text{Minimize } f(x) \\ & x \in \mathbb{R}^n \\ & \text{onde } f : \mathbb{R}^n \to \mathbb{R} \end{array}$$

3.2 O Método de Newton em uma Variável

Nesta seção nos concentramos na solução do problema P_2 em uma variável, ou seja, desejamos encontrar $x \in \mathbb{R}$ tal que $c(x) = 0$, onde $c(x) : \mathbb{R} \to \mathbb{R}$. O método, como o próprio nome sugere, foi proposto por Isaac Newton. O método de Newton

é um processo iterativo de fundamental importância no desenvolvimento de muitos algoritmos de programação não-linear. Seja x_k a solução candidata na iteração k, sendo a solução inicial x_0 definida arbitrariamente. A idéia básica do método de Newton está na aproximação de $c(x)$ com uma série de Taylor de primeira ordem em torno do ponto x_k. Isso nos leva a seguinte aproximação:

$$c(x_{k+1}) = c(x_k) + c'(x_k)(x_{k+1} - x_k)$$

onde $c'(x) = dc/dx$. O método calcula a próxima solução candidata de forma a resolver a aproximação linear. Mais precisamente:

$$c(x_k) + c'(x_k)(x_{k+1} - x_k) = 0$$

$$x_{k+1} = x_k - [c'(x_k)]^{-1} c(x_k)$$

Tipicamente, não se espera que $c(x_{k+1}) = 0$, mas que x_{k+1} defina uma solução candidata de "melhor qualidade" do que aquela definida por x_k. Formalmente, espera-se que:

$$|x_{k+1} - x^*| < |x_k - x^*| \quad e$$

$$|c'(x_{k+1})| < |c'(x_k)|$$

onde x^* é uma solução, i.e., $c(x^*) = 0$.

Uma seqüência de soluções $x_0, x_1, \ldots$ é dita convergente se $\lim_{k\to\infty} |x_k - x^*| = 0$.

Método de Newton em uma variável

Dado um x inicial e uma tolerância $\epsilon > 0$
Repita
  Calcule $c(x)$ e $c'(x)$
  Se $|c(x)| < \epsilon$, retorne x
  $x \leftarrow x - c(x)/c'(x)$
Até que um número máximo de iterações seja atingido

3.2.1 Exemplo

Procuramos um zero para a função $c(x) = -10 - 2x + 4x^2$. A Tabela 3.1 mostra os resultados do método de Newton para $x_0 = 1000$. A Figura 3.1 ilustra o processo iterativo de Newton.

É importante ressaltar que o método de Newton apresenta uma taxa de convergência quadrática. Isso significa, de maneira informal, que o número de dígitos corretos na solução candidata duplica a cada iteração. (O método de Newton apresenta convergência Q-quadrática, ou seja, existe $M \in \mathbb{R}$ tal que $\|x_{k+1} - x^*\|/\|x_k - x^*\|^2 \leqslant M$ para k suficientemente grande se x_0 pertence à região de convergência.) Então qual é o problema com o método de Newton? O método de Newton pode divergir. Isso ocorre quando a solução inicial não é suficientemente próxima da solução ótima. O método de Newton se comporta bem na região próxima da solução, todavia não apresenta convergência global como o método de descenso.

Tabela 3.1: Iterações do método de Newton.

k	x_k	$c(x_k)$	$\lvert x_k - x^* \rvert$
0	1000	3.997990000000000e+006	9.981492189406418e+002
1	500.1262815703926	9.994949375065696e+005	4.982755005110344e+002
2	250.1907039194102	2.498711719029211e+005	2.483399228600520e+002
3	125.2254781755631	6.246523058084260e+004	1.233746971162049e+002
4	62.74799109896568	1.561374556562558e+004	6.089721003960747e+001
5	31.51949620842356	3.900875572514462e+003	2.966871514906535e+001
6	15.92572254113459	9.726631087465399e+002	1.407494148177637e+001
7	8.169595939998821	2.406299994113832e+002	6.318814880640609e+000
8	4.371580213652639	5.769969383029171e+001	2.520799154294426e+000
9	2.621653885664322	1.224896861354670e+001	7.708728263061098e-001
10	1.976061758605505	1.667156778081310e+000	1.252806992472928e-001
11	1.855327609771498	5.830693877868853e-002	4.546550413286443e-003
12	1.850787497645492	8.245047246546733e-005	6.438287280197130e-006
13	1.850781059371159	1.658051473896194e-010	1.294742091317858e-011
14	1.850781059358212	0	2.220446049250313e-016

3.3 O Método de Newton para Minimização em uma Variável

Nesta seção tratamos do problema de encontrar um valor $x \in \mathbb{R}$ tal que $f(x) : \mathbb{R} \to \mathbb{R}$ seja mínimo. O método inicia, como no caso anterior, com uma aproximação da função em torno da solução candidata corrente, x_k, mas desta vez utilizamos uma aproximação de segunda ordem[1].

$$f(x_{k+1}) = f(x_k) + f'(x_k)[x_{k+1} - x_k] + \frac{1}{2} f''(x_k)[x_{k+1} - x_k]^2.$$

Para que x_{k+1} seja o mínimo da aproximação, necessitamos que:

$$\begin{aligned} df/dx_{k+1} = 0 \quad &\Rightarrow \quad f'(x_k) + f''(x_k)[x_{k+1} - x_k] = 0 \\ &\Rightarrow \quad x_{k+1} = x_k - [f''(x_k)]^{-1} f'(x_k). \end{aligned}$$

3.3.1 Exemplo

Encontre um mínimo de $f(x) = \cos(x)$ (conforme Figura 3.2). Se iniciamos o processo iterativo no ponto $x_0 = \pi/4$, o método converge para o ponto $x^* = 0$[2]. Se iniciamos no ponto $x_0 = 3\pi/4$, o método converge para o ponto $x^* = \pi$[3]. O que levou o método de Newton a convergir para um ponto de máximo? Será que o método apresenta uma falha fundamental?

[1] Poderíamos ter adotado uma aproximação de primeira ordem?

[2] Note que $\cos(x^*) = 1$

[3] Observe que $\cos(\pi) = -1$

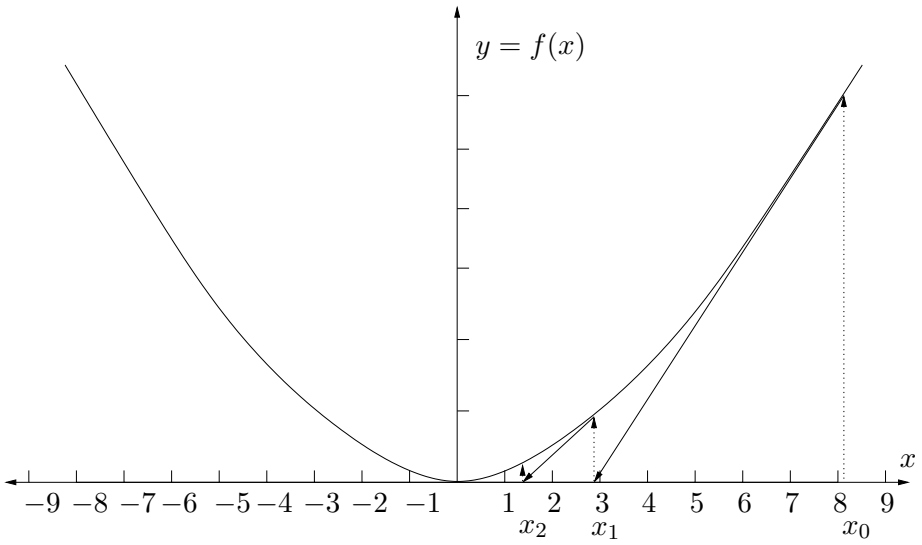


Figura 3.1: Ilustração do processo iterativo de Newton.

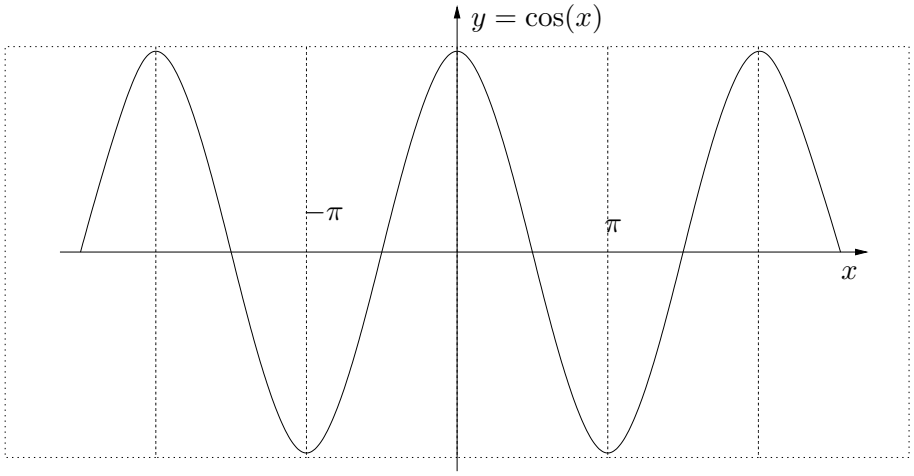


Figura 3.2: Função $\cos(x)$

Note que o método de Newton gera uma seqüência de soluções x_0, x_1, $\ldots, x^*$ que converge para um ponto x^* estacionário, ou seja, $f'(x^*) = 0$. Todavia, x^* não é necessariamente um ponto de mínimo, podendo também definir um ponto de inflexão (e.g., o ponto estacionário da função $f(x) = x^3$) ou de máximo (e.g., o ponto estacionário da função $f(x) = -x^2$). Abaixo relembramos as condições necessárias e suficientes para mínimo local:

Condições Necessárias Para Mínimo Local: $f'(x^*) = 0$ *e* $f''(x^*) \geqslant 0$

Condições Suficientes Para Mínimo Local: $f'(x^*) = 0$ *e* $f''(x^*) > 0$.

3.4 O Método de Newton em Múltiplas Variáveis

Desejamos encontrar um vetor $x = (x_1, \ldots, x_n)$ tal que:

$$c(x) = \begin{bmatrix} c_1(x) \\ \vdots \\ c_m(x) \end{bmatrix} = 0 \tag{3.1}$$

Para simplificar a apresentação, suponha que $m = n$. Da mesma forma que no caso de uma variável, utilizamos uma aproximação linear de $c(x)$ em torno da solução candidata x_k:

$$c(x_{k+1}) = c(x_k) + G(x_k)[x_{k+1} - x_k] \tag{3.2}$$

sendo a matriz $G(x) = \nabla c(x)$ chamada de Jacobiana e definida como segue:

$$G(x) = \begin{bmatrix} \partial c_1/\partial x_1 & \ldots & \partial c_1/\partial x_n \\ \vdots & \ddots & \vdots \\ \partial c_n/\partial x_1 & \ldots & \partial c_n/\partial x_n \end{bmatrix} = \begin{bmatrix} \nabla c_1^T \\ \vdots \\ \nabla c_n^T \end{bmatrix} \tag{3.3}$$

Para encontrar o próximo iterando, fazemos $c(x_k)+G(x_k)[x_{k+1}-x_k] = 0$, resultando no processo iterativo abaixo:

$$x_{k+1} = x_k - G^{-1}(x_k)c(x_k). \tag{3.4}$$

Lembramos que o método de Newton em múltiplas variáveis herda as mesmas propriedades de sua versão monovariada:

- o método apresenta convergência quadrática, desde que o ponto inicial x_0 esteja dentro de sua região de convergência;
- o método pode divergir se uma técnica de globalização não for empregada; e
- a matriz Jacobiana deve ser não-singular para que o método tenha sucesso.

Método de Newton em Várias Variáveis

Dado um x inicial e uma tolerância $\epsilon > 0$
Repita
    Calcule $c(x)$ e $G(x)$
    Se $\|c(x)\| < \epsilon$, retorne x
    Encontre uma solução para $G(x)\Delta x = -c(x)$
    Faça $x \leftarrow x + \Delta x$
Até que um número máximo de iterações seja atingido

Enfatizamos que a iteração Newton, conforme Equação (3.4), não é realizada através do cálculo direto da inversa de $G(x)$. Fazemos uso de algoritmos de solução de sistemas de equações lineares para resolver o sistema $G(x)\Delta x = -c(x)$. Um forma robusta para resolvê-lo se dá através da decomposição LU, a qual decompõe G em duas matrizes triangulares L (*triangular inferior*) e U (*triangular superior*), de maneira que $G = LU$. Uma vez calculados os fatores L e U, a solução de (3.4) pode ser obtida com duas retrosubstituições. Quando a matriz G é positiva definida, o que ocorre no caso do método de Newton estar sendo utilizado como ferramenta para encontrar um ponto estacionário de uma função f, é recomendável realizar uma fatoração Cholesky. Esta encontra uma matriz triangular inferior L de forma que $G = LL^T$. Algoritmos eficientes e robustos podem ser encontrados na literatura para executar ambos os tipos de fatoração.

3.5 Minimização Irrestrita

Aqui a atenção se volta para o problema de minimização irrestrita em múltiplas variáveis. Desejamos encontrar um vetor $x = (x_1, \ldots, x_n)$ tal que o valor da função $f(x)$ seja mínimo. Vamos aproximar $f(x)$ em torno da solução candidata corrente x_k como segue:

$$f(x_{k+1}) = f(x_k) + \nabla f(x_k)^T[x_{k+1} - x_k] + \frac{1}{2}[x_{k+1} - x_k]^T \nabla^2 f(x_k)[x_{k+1} - x_k]$$

Fazendo $x_{k+1} = x^+$, $x_k = x$, $\nabla f(x_k) = g$, $\nabla^2 f(x_k) = H$ e $p = [x_{k+1} - x_k]$, a aproximação pode ser reescrita como segue:

$$f(x^+) = f(x) + g^T p + \frac{1}{2} p^T H p.$$

Da mesma forma que no caso monovariado, podemos encontrar um ponto estacionário para a aproximação forçando x^+ a satisfazer a condição necessária de primeira ordem. Formalmente:

$$\nabla f(x^+) = 0 \Rightarrow g + Hp = 0 \Rightarrow p = -H^{-1} g.$$

Portanto, o processo iterativo pode ser expresso como segue:

$$x_{k+1} = x_k - [\nabla^2 f(x_k)]^{-1} \nabla f(x_k).$$

Uma vez que o método de Newton se baseia na busca de um ponto para o qual o gradiente de $f(x)$ é nulo, não há garantia de que o processo iterativo convirja para um ponto de mínimo, podendo divergir ou ser atraído para um ponto de inflexão ou de máximo. Lembramos que as condições necessárias e suficientes (*minimização de funções*) para que x_k seja um mínimo local são:

Condições Necessárias: $\nabla f(x_k) = 0$ e $\nabla^2 f(x_k) \geqslant 0$ (positiva semi-definida).

Condições Suficientes: $\nabla f(x_k) = 0$ e $\nabla^2 f(x_k) > 0$ (positiva definida).

3.6 Convergência

Algoritmos não-iterativos, como o método de Gauss para solução de sistemas de equações lineares, têm custo computacional bem-definido. O número de operações de ponto-flutuante é finito e pode ser expresso como uma função polinomial das dimensões do problema. O comportamento de algoritmos iterativos, por outro lado, é mais difícil de ser medido. O número de iterações depende do problema e do ponto inicial. A eficiência de um algoritmo iterativo é medida em termos de um limite superior de iterações necessárias para se atingir uma solução com certa precisão. Tais limites são obtidos por meio de uma análise de convergência.

3.6.1 Convergência Linear

Uma seqüência x_k com limite x^* é linearmente convergente se existe uma constante $c \in (0, 1)$ tal que:

$$|x_k - x^*| \leqslant c|x_{k-1} - x^*| \tag{3.5}$$

para k suficientemente grande. Convergência linear é algumas vezes definida como segue. A seqüência x_k com limite x^* converge R-linearmente se existe um valor M e $c \in (0, 1)$ tal que:

$$|x_k - x^*| \leqslant Mc^k \tag{3.6}$$

para k suficientemente grande.

3.6.2 Convergência Quadrática

Uma seqüência x_k com limite x^* converge quadraticamente se existe uma constante $c > 0$ tal que:

$$|x_k - x^*| \leqslant c|x_{k-1} - x^*|^2 \tag{3.7}$$

para k suficientemente grande. A seqüência $x_k = 1 + (\frac{1}{2})^{2^k}$ converge quadraticamente para $x^* = 1$ pois:

$$|x_{k+1} - x^*| = \left(\frac{1}{2}\right)^{2^{k+1}} = \left[\left(\frac{1}{2}\right)^{2^k}\right]^2 = |x_k - x^*|^2. \tag{3.8}$$

Definindo $r_k = -\log_{10}[|x_k - x^*|/|x^*|]$, quando $|x^*| \neq 0$, e dividindo por $|x^*|$ podemos reescrever (3.7) como

$$r_k \geqslant 2r_{k-1} - \log_{10}(|x^* c|). \tag{3.9}$$

Uma vez que $|x_k - x^*| \to 0$, $r_k \to +\infty$ e eventualmente o primeiro termo da parte à direita da desigualdade dominará o segundo. r_k é aproximadamente o número de dígitos corretos na iteração k, portanto o número de dígitos corretos praticamente dobra a cada iteração.

3.6.3 Convergência do Método de Newton

O método de Newton converge quadraticamente se $G(x^*)$ tem posto completo e se x_0 é suficientemente próximo de x^*. Para o caso unidimensional, a ser estudado nesta seção, a convergência quadrática é obtida se $f'(x^*) \neq 0$. Seja $I = [x^* - \delta, x^* + \delta]$ um intervalo em torno da solução x^* dentro do qual as seguintes condições são satisfeitas:

i. existe uma constante $m > 0$ tal que $|f'(x)| \geqslant m$ para todo $x \in I$; e

ii. existe uma constante $L > 0$ tal que $|f'(x) - f'(y)| \leqslant L|x - y|$ para todo $x, y \in I$.

Vamos mostrar que se $|x_k - x^*| \leqslant \delta$, então

$$|x_{k+1} - x^*| \leqslant \frac{L}{2m}|x_k - x^*|^2 \tag{3.10}$$

Em outras palavras, a desigualdade (3.7) é satisfeita com $c = L/(2m)$, garantindo dessa forma convergência quadrática para x^*. Denotando x_{k+1} por x^+ e x_k por x, temos que

$$\begin{aligned} x+ = x - \frac{f(x)}{f'(x)} \Rightarrow |x^+ - x^*| &= |x - \frac{f(x)}{f'(x)} - x^*| \\ &= \frac{|-f(x) - f'(x)(x^* - x)|}{|f'(x)|} \\ &= \frac{|f(x^*) - f(x) - f'(x)(x^* - x)|}{|f'(x)|} \end{aligned}$$

Lembre-se que $f(x^*) = 0$. Nós assumimos que $|f'(x)| \geqslant m$ e, portanto,

$$|x^+ - x^*| \leqslant \frac{|f(x^*) - f(x) - f'(x)(x^* - x)|}{m} \tag{3.11}$$

Para limitar o numerador, podemos proceder como segue

$$\begin{aligned} |f(x^*) - f(x) - f'(x)(x^* - x)| &= |\int_x^{x^*} (f'(u) - f'(x))du| \\ &\leqslant \int_x^{x^*} |f'(u) - f'(x)|du \\ &\leqslant L\int_x^{x^*} |u - x|du \\ &= L\frac{|x^* - x|^2}{2} \end{aligned} \tag{3.12}$$

Juntando (3.11) e (3.12) obtemos $|x^+ - x^*| \leqslant \frac{L}{2m}|x^* - x|^2$ o que prova (3.10). A desigualdade (3.10) por si só não garante que $|x_{k+1} - x^*| \to 0$. Entretanto, se assumirmos que $|x_0 - x^*| \leqslant \delta$ e $\delta \leqslant m/L$, então convergência quadrática segue imediatamente. Uma vez que $|x_0 - x^*| \leqslant \delta$, podemos aplicar (3.10) e obter:

$$|x_1 - x^*| \leqslant \frac{L}{2m}|x_0 - x^*|^2 \leqslant \frac{L}{2m}\delta^2 \leqslant \delta/2.$$

Uma vez que $|x_1 - x^*| \leqslant \delta$, podemos aplicar (3.10) e obter a desigualdade:

$$|x_2 - x^*| \leqslant \frac{L}{2m}|x_1 - x^*|^2 \leqslant \frac{L}{2m}\delta^2/4 \leqslant \delta/8.$$

Repetindo este processo, concluímos que:

$$|x_{k+1} - x^*| \leqslant \frac{L}{2m}|x_k - x^*|^2 \leqslant 2\left(\frac{1}{4}\right)^{2^k}\delta.$$

Segundo o desenvolvimento acima, se $f(x^*) = 0$ e f' é uma função contínua, então é razoável assumir que as condições acima são satisfeitas para algum δ, m e L. Na prática, entretanto, é difícil encontrarmos valores para δ, m e L, portanto o resultado de convergência não nos oferece ajuda no sentido de escolhermos o ponto inicial x_0, mesmo porque isto demandaria conhecimento antecipado da existência de uma raiz. O resultado nos diz que se x_0 é suficientemente próximo de x^*, então o método de Newton converge com uma taxa quadrática.

3.7 Métodos de Região de Confiança

Como visto acima, o método de Newton puro para minimização de funções encontra $p = x - x_k$ que minimiza uma aproximação de f em torno de x_k dada pela expansão de Taylor de segunda ordem:

$$f_k(p) = f(x_k) + \nabla f(x_k)^T p + \frac{1}{2}p^T \nabla^2 f(x_k) p.$$

Sabemos que $f_k(p)$ é uma boa aproximação de $f(x_k + p)$ quando p está dentro de uma vizinhança próxima de zero, mas a dificuldade é que na minimização de $f_k(p)$ podemos obter um passo fora desta região. Se torna portanto relevante considerar um passo de Newton restrito p_k obtido por meio da minimização de $f_k(p)$ dentro de uma região pequena na vizinhança do zero, chamada região de confiança (*trust region*):

$$\begin{array}{rcll} p_k & = & \underset{\|p\| \leqslant \gamma_k}{\text{Argmin}} & f_k(p) \end{array} \tag{3.13}$$

onde γ_k é um escalar positivo. Explorando o fato de que o problema (3.13) tem apenas uma restrição, uma solução aproximada pode ser obtida rapidamente.

É importante enfatizar que mesmo quando $\nabla^2 f(x_k)$ não é positiva definida ou até mesmo quando a direção Newton não é uma direção de descenso, o passo de Newton

restrito p_k reduz o custo da função, desde que $\nabla f(x_k) \neq 0$ e γ_k seja suficientemente pequeno. Note que para todo p tal que $\|p\| \leqslant \gamma_k$,

$$f(x_k + p) = f_k(p) + o(\gamma_k^2)$$

e portanto

$$\begin{aligned} f(x_k + p_k) &= f_k(p_k) + o(\gamma_k^2) \\ &= f(x_k) + Min_{\|p\| \leqslant \gamma_k}\{\nabla f(x_k)^T p + \frac{1}{2} p^T \nabla^2 f(x_k) p\} + o(\gamma_k^2). \end{aligned}$$

Denotando

$$\tilde{p}_k = -\frac{\nabla f(x_k)}{\|\nabla f(x_k)\|}\gamma_k,$$

temos

$$\begin{aligned} f(x_k + p_k) &\leqslant f(x_k) + \nabla f(x_k)^T \tilde{p}_k + \frac{1}{2}\tilde{p}_k^T \nabla^2 f(x_k)\tilde{p}_k\} + o(\gamma_k^2) \\ &= f(x_k) - \gamma_k \|\nabla f(x_k)\| + \frac{\gamma_k^2}{2\|\nabla f(x_k)\|^2}\nabla f(x_k)^T \nabla^2 f(x_k)\nabla f(x_k) + o(\gamma_k^2) \end{aligned}$$

Para γ_k suficientemente pequeno o termo negativo $-\gamma_k\|\nabla f(x_k)\|$ domina os últimos dois termos, o que nos leva a concluir que: $f(x_{k+1}) < f(x_k)$. Podemos também observar que uma redução nos custos pode ser obtida mesmo quando $\nabla f(x_k) = 0$ desde que γ_k seja suficientemente pequeno e f tenha uma direção de curvatura negativa no ponto x_k, o que significa dizer que $\nabla^2 f(x_k)$ não é positiva semidefinida. Portanto, o procedimento só falha em reduzir o custo quando $\nabla f(x_k) = 0$ e $\nabla^2 f(x_k)$ é positiva semidefinida, que corresponde a dizer que x_k satisfaz as condições necessárias de segunda ordem.

Os desenvolvimentos acima nos levam a considerar passos da forma

$$x_{k+1} = x_k + p_k,$$

onde p_k é o passo de Newton restrito com respeito a um escalar γ_k. A chave para a convergência rápida do método de Newton com região de confiança está em ajustar dinamicamente o parâmetro γ_k, garantindo redução significativa e se comportando como o método de Newton puro próximo de um mínimo local não singular[4]. Uma forma razoável de ajustar o valor inicial de γ_k é incrementar o valor quando o método parece estar gerando reduções no valor de f e reduzi-lo caso contrário. O progresso em direção a um ótimo local pode ser medido com base na razão entre a redução antecipada e a redução efetivamente observada:

$$r_k = \frac{f(x_k) - f(x_{k+1})}{f(x_k) - f_k(p_k)}.$$

Aumenta-se o valor de γ se esta razão está próxima ou acima da unidade, e reduzimos o seu valor caso contrário. Usando como parâmetros $0 < \sigma_1 \leqslant \sigma_2 \leqslant 1$ e $0 < \beta_1 < 1 < \beta_2$, um algoritmo pode ser proposto:

[4]Um mínimo local x é não singular se satisfaz as condições suficientes de segunda ordem, ou seja, $\nabla f(x) = 0$ e $\nabla^2 f(x) > 0$

Passo 1: Encontre:

$$p_k = \begin{array}{ll} \text{Argmin} & f_k(p) \\ \|p\| \leqslant \gamma_k & \end{array}$$

Se $f_k(p_k) = f(x_k)$ pare (x_k satisfaz as condições necessárias de primeira e segunda ordem).

Passo 2: Se $f(x_k + p_k) < f(x_k)$ defina $x_{k+1} = x_k + p_k$, calcule

$$r_k = \frac{f(x_k) - f(x_{k+1})}{f(x_k) - f_k(p_k)}$$

e vá para o Passo 3, caso contrário faça $\gamma_k = \beta_1 \|p_k\|$ e vá para o Passo 1.

Passo 3: Faça

$$\gamma_{k+1} = \begin{cases} \beta_1 \|p_k\| & \text{se } r_k < \sigma_1 \\ \beta_2 \gamma_k & \text{se } \sigma_2 \leqslant r_k \text{ e } \|p_k\| = \gamma_k \\ \gamma_k & \text{caso contrário} \end{cases}$$

Volte ao Passo 1.

Se f é duas vezes diferenciável então é possível mostrar que o algoritmo acima converge, i.e., a seqüência $\{x_k\}$ é limitada e existe um ponto limite que satisfaz as condições necessárias de segunda ordem. Além disso, se $\{x_k\}$ converge para um mínimo local x^* não singular, então o método tem a mesma taxa de convergência do método de Newton puro.

3.8 Otimização de Funções Sujeito a Igualdades

Até aqui vimos como o método de Newton pode ser empregado para encontrar uma solução simultânea de um sistema $c(x)$ de equações não-lineares e também encontrar um ótimo local para uma função $f(x)$ (i.e., solução que satisfaz as condições necessárias de primeira ordem). Aqui vamos combinar estes dois problemas e estender o método de Newton. Esta seção pode ser omitida sem perda de informação já que a otimização restrita será objeto de estudo detalhado em capítulos futuros. Considere o problema de encontrar um vetor x que minimiza uma função $f(x)$ e ao mesmo tempo é solução simultânea de um sistema de equações:

$$\begin{array}{ll} \text{Minimize} & f(x) \\ \text{Sujeito a :} & \\ & c(x) = 0 \end{array}$$

onde $x \in \mathbb{R}^n$, $f : \mathbb{R}^n \to \mathbb{R}$ e $c : \mathbb{R}^n \to \mathbb{R}^m$ com $m < n$. Pela teoria clássica de otimização, podemos tomar o Lagrangeano:

$$\mathcal{L}(x, \lambda) = f(x) - \lambda^T c(x) = f(x) - \sum_{i=1}^{m} \lambda_i c_i(x)$$

onde $\lambda \in \mathbb{R}^m$ é o vetor com os multiplicadores de Lagrange. Da mesma forma que o problema de minimização irrestrita, para um par solução-multiplicador (x^*, λ^*) definir uma solução localmente ótima a derivada deve ser nula com respeito a x e λ, portanto:

$$\nabla_x \mathcal{L}(x^*, \lambda^*) = 0 \tag{3.14}$$
$$\nabla_\lambda \mathcal{L}(x^*, \lambda^*) = 0 \tag{3.15}$$

Calculando as derivadas conforme (3.14) e (3.15), obtemos:

$$\begin{aligned} \nabla_x \mathcal{L} &= \nabla f(x) - \sum_{i=1}^{m} \lambda_i \nabla c_i(x) \\ &= \nabla f(x) - \nabla c(x)^T \lambda \qquad (3.16) \\ \nabla_\lambda \mathcal{L} &= -c(x) \qquad (3.17) \end{aligned}$$

onde $\nabla c(x)$ é o Jacobiano de c. Note que as condições (3.14) e (3.15) podem ser satisfeitas por um par (x^*, λ^*) que induz um mínimo local, máximo local ou simplesmente um ponto estacionário. Da mesma forma que em otimização irrestrita, podemos desenvolver condições necessárias de segunda ordem. Primeiramente, vamos calcular a Hessiana do Lagrangeano com respeito a x:

$$\begin{aligned} H &= \nabla^2_{xx} \mathcal{L} \\ &= \nabla^2 f(x) - \sum_{i=1}^{m} \lambda_i \nabla^2 c_i(x) \qquad (3.18) \end{aligned}$$

Então pode ser demonstrado que condições necessárias (3.19) e suficientes (3.20) para que o par (x^*, λ^*) seja um mínimo local são respectivamente:

$$u^T H u \geqslant 0 \tag{3.19}$$
$$u^T H u > 0 \tag{3.20}$$

para todo u tangente ao espaço de restrições no ponto x^*, ou seja, para todo $u \in \{v : \nabla c_i(x^*)^T v = 0, i = 1, \ldots, m\}$. Enquanto que as condições de segunda ordem em otimização irrestrita exigem que a curvatura de f seja positiva em todas as direções, sob restrições de igualdade a curvatura deve ser positiva apenas nas direções factíveis (tangentes ao espaço de soluções factíveis no ponto x^*).

De forma análoga à aplicação do método de Newton para minimização de funções, podemos desenvolver um método de Newton para encontrar um par (x^*, λ^*) que satisfaz as condições necessárias de primeira ordem. Primeiro, aproximamos a função $\nabla_x \mathcal{L}$ com um aproximador linear em torno de $(\bar{x}, \bar{\lambda})$:

$$\begin{aligned} \nabla_x \mathcal{L}(x, \lambda) &= \nabla_x \mathcal{L}(\bar{x}, \bar{\lambda}) + \nabla^2 f(\bar{x})(x - \bar{x}) \\ &\quad - \sum_{i=1}^{m} \lambda_i \nabla^2 c_i(\bar{x})(x - \bar{x}) - \sum_{i=1}^{m} \nabla c_i(\bar{x})(\lambda_i - \bar{\lambda}_i) \\ &= \nabla_x \mathcal{L}(\bar{x}, \bar{\lambda}) + [\nabla^2 f(\bar{x}) - \sum_{i=1}^{m} \lambda_i \nabla^2 c_i(\bar{x})](x - \bar{x}) - \nabla c(\bar{x})^T (\lambda - \bar{\lambda}) \\ &= \nabla f(\bar{x}) - \nabla c(\bar{x})^T \bar{\lambda} + H(x - \bar{x}) - \nabla c(\bar{x})^T (\lambda - \bar{\lambda}) \\ &= \nabla f(\bar{x}) + H(x - \bar{x}) - \nabla c(\bar{x})^T \lambda \qquad (3.21) \end{aligned}$$

Segundo, aproximamos a função $\nabla_\lambda \mathcal{L}$ com um aproximador linear, obtendo:

$$\nabla_\lambda \mathcal{L}(x,\lambda) \;=\; -c(\bar{x}) - \nabla c(\bar{x})(x-\bar{x}) \tag{3.22}$$

Logo, dado um par (x_k, λ_k), o próximo iterando (x_{k+1}, λ_{k+1}) do método de Newton é obtido resolvendo um sistema de equações lineares:

$$\left\{ \begin{array}{rcl} \nabla_x \mathcal{L}(x_{k+1},\lambda_{k+1}) &=& 0 \\ \nabla_\lambda \mathcal{L}(x_{k+1},\lambda_{k+1}) &=& 0 \end{array} \right. \Leftrightarrow \left\{ \begin{array}{rcl} \nabla f(x_k) + H(x_{k+1}-x_k) - \nabla c(x_k)^T \lambda_{k+1} &=& 0 \\ -c(x_k) - \nabla c(x_k)(x_{k+1}-x_k) &=& 0 \end{array} \right.$$

ou, de forma mais compacta,

$$\left[\begin{array}{cc} H & -\nabla c(x_k)^T \\ -\nabla c(x_k) & 0 \end{array} \right] \left[\begin{array}{c} x_{k+1} \\ \lambda_{k+1} \end{array} \right] = \left[\begin{array}{c} Hx_k - \nabla f(x_k) \\ c(x_k) - \nabla c(x_k) x_k \end{array} \right] \tag{3.23}$$

3.9 Referências

Para um tratamento aprofundado e rigoroso do método de Newton recomendamos o livro texto de Bertsekas [9] sobre programação não-linear. Para problemas de minimização de funções, os métodos quasi-Newton diferem do método de Newton no sentido que os primeiros não calculam a matriz Hessiana $\nabla^2 f(x_k)$ a cada iterando x_k. Em vez disso, os métodos quasi-Newton utilizam uma aproximação B_k de $\nabla^2 f_k$ que é atualizada com base na seqüência de pares $(x_0, f_0), \ldots, (x_k, f_k)$. Aqueles interessados em métodos quasi-Newton e os que desejam resolver problemas de grande porte devem obrigatoriamente ler o *survey* de Dennis e Moré [16]. Este artigo apresenta de forma clara e sucinta as regras de atualização de B_k propostas por: Broyden (B); Davidon, Fletcher e Powell (DFP); e Broyden, Fletcher, Goldfarb e Shanno (BFGS).

3.10 Exercícios

EX 3.1: Seja $c(x) = \arctan(x)$. Encontre a região de convergência e a de divergência do método de Newton.

EX 3.2: Considere o sistema de equações não-lineares a seguir:

$$\begin{array}{rcl} \log(x_1^2 + 2x_2^2 + 1) - \dfrac{1}{2} &=& 0 \\ x_2 - x_1^2 + 0.2 &=& 0 \end{array}$$

Implemente em Matlab o algoritmo de Newton para resolver o sistema acima. Para cada um dos seguintes iterandos iniciais, indique se o algoritmo converge e, em caso afirmativo, o ponto para o qual ele converge: $x_0 = (1,1)$; $x_0 = (-1,1)$; $x_0 = (1,-1)$; e $x_0 = (-1,-1)$.

EX 3.3: Considere uma transformação linear inversível $x = Sy$. Escreva o método de Newton no espaço da variável y e mostre que este gera uma seqüência $y_k = S^{-1}x_k$, onde $\{x_k\}$ é a seqüência gerada pelo método de Newton no espaço da variável x.

EX 3.4: Seja $A \in \mathbb{R}^{m\times n}$ uma matriz, $rank(A) = m$, $c \in \mathbb{R}^n$. Considere o problema abaixo:

$$\begin{array}{lll} P: & \text{Minimize} & \|x-c\|^2 \\ & \text{Sujeito a :} & \\ & & Ax = 0 \\ & & x \in \mathbb{R}^n \end{array}$$

Transforme P em um problema de otimização irrestrita, permitindo o emprego do algoritmo de Newton. Podemos garantir que um ponto estacionário define um ótimo local? Podemos afirmar que um ótimo local é um ótimo global?

EX 3.5: Embora as aplicações do método de Newton em Cálculo sejam bastante antigas, só recentemente descobriu-se que sua extensão para o plano complexo pode gerar fractais. Em 1879, Caley formulou a seguinte questão: dado um ponto inicial z_0, para qual das raízes de uma função complexa $f(z)$ o método de Newton converge?

Seja $f(x) = z^3 + 2$. Encontre as três raízes desta função. Produza uma figura seguindo o procedimento abaixo:

1) Se restrinja ao quadrado $[-2, 2] \times [-2, 2]$ do plano complexo.
2) Divida esta região do plano em pequenos subquadrados.
3) Execute o método de Newton a partir do número complexo definido pelo subquadrado.
4) Pinte o subquadrado com a cor vermelha, amarela ou azul, dependendo da raiz para a qual o método convergiu.

EX 3.6 [9]: Considere o problema de otimização irrestrita onde β é um parâmetro:

$$\underset{x,y}{\text{Minimize}} \quad f(x,y) = x^2 + y^2 + \beta xy + x + 2y$$

Quais pontos são estacionários? Quais desses pontos definem mínimos locais? Quais desses pontos são mínimos globais?

EX 3.7 [9]: Para cada uma das funções abaixo, encontre os pontos estacionários. Quais desses pontos são mínimos ou máximos locais? Quais desses pontos são mínimos ou máximos globais?

1) $f(x,y) = (x^2-4)^2 + y^2$
2) $f(x,y) = (y-x^2)^2 - x^2$
3) $f(x,y) = \frac{1}{2}x^2 + x\cos(y)$

EX 3.8 [9]: Desejamos encontrar o ponto x em um plano cuja soma das distâncias ponderadas a partir de um conjunto de pontos $y_1, \ldots, y_m$ seja minimizada. Matematicamente, o problema é:

$$\underset{x\in\mathbb{R}^n}{\text{Minimize}} \quad \sum_{k=1}^{m} w_k \|x - y_k\|$$

onde $w_1, \ldots, w_m$ são escalares positivos. Mostre que existe um ótimo global para o problema e que este pode ser realizado através do modelo mecânico dado na Fig. 3.3. Esta solução ótima global é única?

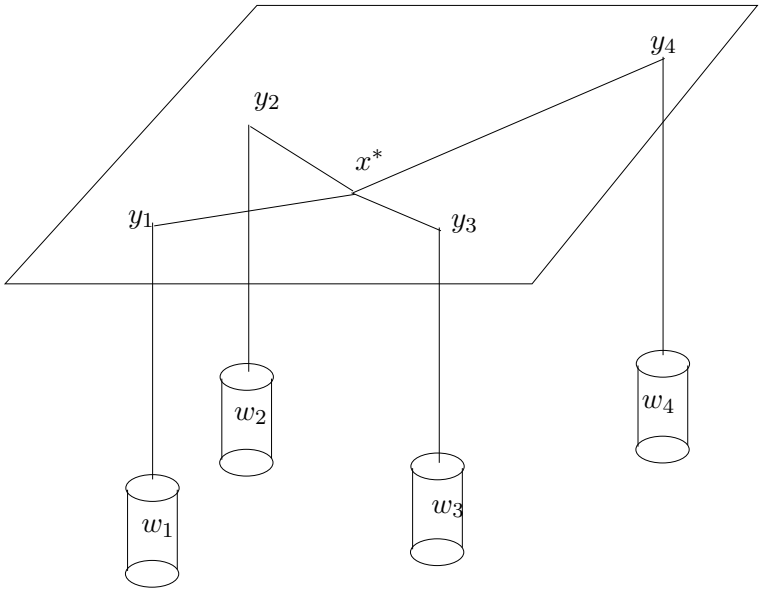


Figura 3.3: Modelo mecânico. O modelo consiste de um tabuleiro com um furo perfurado a cada ponto y_k. Através da cada buraco, um fio é passado com o correspondente peso w_k preso a sua ponta. Os outros extremos dos fios são amarrados com um nó. Na ausência de fricção e fios entrelaçados, as forças atingem um equilíbrio no ponto do nó quando este está localizado na solução ótima x^*.

Capítulo 4

Otimização Não-Diferenciável

Nos capítulos anteriores nos concentramos nos problemas de encontrar um ótimo local de uma função $f(x)$ e de resolver um sistema de equações não-lineares $c(x) = 0$. Em ambas as situações assumimos que as funções f e c são contínuas e diferenciáveis. Isto permitiu utilizar aproximações de primeira e segunda ordem com base na expansão de Taylor, bem como o desenvolvimento de algoritmos que fazem uso do gradiente (no caso de minização de $f(x)$) ou da matriz Jacobiana (no caso de encontrar raiz para $c(x)$). Aqui vamos tratar o problema de encontrar um mínimo para uma função $f(x)$ qualquer que pode ser descontínua e não diferenciável. Logo, métodos inspirados na expansão de Taylor não são aplicáveis. Apresentaremos de forma sucinta o Algoritmo Genético e *Simulated Annealing* que podem ser empregados para resolver tais problemas.

4.1 Otimização *Black-Box*

Neste capítulo focamos nossa atenção no problema geral de otimizaçção irrestrita, sendo este expresso em programação matemática como:

$$\begin{aligned} P: \quad & \text{Minimize} \quad f(x) \\ & x \in \mathbb{R}^n \\ & \qquad\qquad \text{onde nada se assume sobre } f : \mathbb{R}^n \to \mathbb{R} \end{aligned}$$

Uma gama expressiva de problemas práticos e teóricos pode ser reduzida a P. Em particular, f pode representar:

- O custo de produção para um dado conjunto de parâmetros que representam indicadores econômicos, custos de matéria-prima e planejamento da produção, podendo este custo ser estimado por meio de simulação computacional.
- O valor da função objetivo de um problema de otimização, onde $f(x) = \infty$ se x é infactível.
- O prêmio recebido em um certo jogo quando o participante segue a estratégia induzida por x.

O termo *otimização não-diferenciável* tem suas raízes no fato que para uma função f qualquer o gradiente pode não estar definido, portanto pode não existir informação

local quanto à direção de descenso. Duas técnicas para otimização não-diferenciável são:

- **Algoritmo Genético (AG).** O AG tem suas raízes na Teoria da Evolução de Charles Darwin, podendo este ser visto como um mecanismo de evolução simulada.
- **Simulated Annealing (SA).** Se inspira no processo físico de obtenção de estruturas cristalinas através do super-aquecimento e resfriamento lento da substância, o que leva o sistema a atingir o nível mais baixo de energia possível.

4.2 Algoritmo Genético (AG)

4.2.1 Genética e Evolução

A percepção de que certas características são hereditárias (i.e., transmitidas geneticamente) data de várias décadas passadas. A Teoria da Evolução [14] é considerada um dos maiores avanços científicos até esta data. Ela tem implicações em várias áreas naturais, científicas e sociais como, por exemplo, Biologia, Matemática, Física e Sociologia.

4.2.2 Adaptação Biológica

Estudaremos maneiras pelas quais o processo de evolução natural pode ser empregado para se "evoluir" soluções computacionais (Algoritmo Genético) e até mesmo programas ou algoritmos (Programação Genética) para problemas interessantes.

A propriedade de adaptação é descrita pela seguinte expressão:

Adaptação = Variedade + Hereditariedade + Seleção

1) Variedade: Se refere a como os indivíduos diferem uns dos outros, portanto só pode ocorrer se existirem múltiplos indivíduos, implicando em paralelismo e multiplicidade espacial (*População*).

2) Hereditariedade: É uma forma de persistência temporal que se propaga de pai para filho, um processo iterativo que se repete (*Processo Iterativo*).

3) Seleção Natural: A capacidade de reprodução é exponencial, extremamente maior do que a quantidade de recursos disponíveis (limitada). Surge, portanto, a frase "a sobrevivência dos reprodutores". Levando em conta esta capacidade exponencial, os seres humanos formam um grupo seleto de indivíduos (*Seleção*).

A frase mais conhecida é "a sobrevivência do mais dotado". Entretanto, a teoria diz que o que importa é a reprodução, o que implica a frase "a sobrevivência do reprodutor". Segundo a teoria da evolução, as características de uma pessoa (cor dos olhos, alta, inteligente, etc.) só importam se aumentam a capacidade de reprodução. A evolução das espécies pode levar a processos cíclicos como, por exemplo, a dinâmica entre leões e gazelas. Se os leões se tornam mais perspicazes ao caçarem gazelas, então as gazelas sobreviventes tendem a ser mais rápidas, que por sua vez vão gerar descendentes mais ágeis, dificultando a caça dos leões.

4.2.3 Hereditariedade com Evolução Simulada

Da mesma forma que em sistemas biológicos, as equações de adaptação biológica podem ser utilizadas para "evoluir" soluções de um problema (Algoritmo Genético) e até mesmo algoritmos (Programação Genética). John Holland [28] foi um dos primeiros a propor a simulação do processo de evolução natural para resolver problemas diversos, tais como problemas de otimização e de aprendizagem de máquina. Ele concebeu o termo Algoritmo Genético (AG) e foi um dos grandes impulsionadores da área que hoje é conhecida como *Computação Evolucionária*.

O algoritmo genético representa soluções como estruturas semelhantes ao DNA, as quais expressam alguma forma de aptidão. Isto está inspirado na Natureza, onde estruturas de DNA aparecem em organismos com capacidade de se acasalarem, e onde haja mutação e *cross-over* do material genético. Estes princípios fundamentais foram apresentados através da teoria de Darwin, mais tarde complementada pelos trabalhos de Gregor Mendel (pai da Genética) e posteriormente enriquecidos pela descoberta da estrutura de dupla hélice do DNA. Estes últimos trabalhos mostram que as características transmitidas são discretas, não são contínuas como anteriormente assumido. Mendel mostrou em seus experimentos que, se controlarmos o ambiente, as características são bem distintas (e.g., flores altas ou baixas). Conclusão:

> *A linguagem da natureza é um alfabeto discreto.*

O AG expressa uma solução como uma cadeia de símbolos (usualmente 0s e 1s) de tamanho fixo, da mesma forma que o DNA codifica as características de um indivíduo. É necessário a existência de uma função de aptidão (*fitness function*) que mapeie a cadeia em uma forma útil. Por exemplo, a cadeia pode representar:

- um vetor com as variáveis de uma função f que se deseja minimizar; ou
- uma estratégia para competir em um jogo.

4.2.4 Popularidade do Algoritmo Genético

A popularidade do algoritmo genético se dá em função de três propriedades:

1) Robustez: Evolução natural é tida como um método bem-sucedido e robusto no meio biológico.

2) Generalidade: AGs são capazes de tratar problemas complexos, que possuem um número grande de componentes e cujas contribuições para o todo não são bem entendidas.

3) Paralelização: AGs são naturalmente paralelos; podem ser implementados em redes de computadores (máquinas assíncronos) e computadores paralelos (máquinas síncronos).

Antes de procedermos à descrição dos passos do Algoritmo Genético, assumiremos o seguinte:

- O problema de otimização a ser resolvido é de maximização, em vez de minimização. Para transformar minimização em maximização, basta substituir o operador *min* por *max* e $f(x)$ por $-f(x)$.

- A função objetivo ($f(x)$ se maximização, $-f(x)$ se minimização) será utilizada como função de aptidão (*fitness function*, f_i). Portanto, quanto mais alto o valor da *fitness function* (f_i) melhor a qualidade da respectiva solução. Assumiremos também que a *fitness function* assume um valor não negativo. Isso pode ser contornado através da adição de uma constante, i.e., seja $m = Min\{f_i(x) : x$ pertence à população$\}$, então basta substituir $f_i(x)$ por $f_i(x) - m$ para que a *fitness function* se torne não negativa.

4.2.5 Algoritmo Genético em Detalhes

O algoritmo genético pode ser descrito como uma função $AG(f_i, f_t, p, r, m)$ cujos parâmetros de entrada definem um problema a ser resolvido. Abaixo segue o pseudo-código.

Algoritmo $AG(f_i, f_t, p, r, m)$

Parâmetros de entrada:
f_i - *fitness function*
f_t - *fitness threshold* (condição de parada)
p - tamanho da população
r - fração da população a ser substituída por *cross-over*
m - taxa de mutação

Inicialize a população P: gere p soluções candidatas aleatoriamente
Avalie os elemento de P: para cada $x \in P$, calcule $f_i(x)$

Enquanto $\max\{f_i(x) : x \in P\} < f_t$, execute:
- Seja P' a nova população, iniciando com $P' = \emptyset$
- Selecione probabilisticamente $(1 - r)p$ elementos de P e adicione a P', sendo a probabilidade de $x \in P$ ser selecionado dada por:
  $$Pr(x) = f_i(x) / \sum_{x_j \in P} f_i(x_j)$$
- Probabilisticamente selecione $rp/2$ pares de elementos de P, de acordo com a distribuição dada por $Pr(x)$.
- Para cada par (x_i, x_j) execute *cross-over* e adicione os "filhos" a P'.
- Execute mutação de mp elementos de P'
- Faça $P \leftarrow P'$
- Calcule $f_i(x)$ para todo $x \in P$

4.2.6 Operador Genético *Cross-Over*

Os operadores genéticos determinam como os membros de uma população são combinados, ou sofrem mutação, de forma a gerar novos candidatos. Na Figura 4.1 é ilustrado o operador *cross-over* de um ponto, enquanto que o operador *cross-over* de dois pontos é exemplificado na Figura 4.2. O operador de *cross-over* uniforme consistem em utilizar uma máscara cujos bits são selecionados aleatoriamente com probabilidade uniforme. Já o operador de mutação simplesmente troca um ou mais bits de um cromossomo de 0 para 1 (ou vice-versa). Os bits são escolhidos aleatoriamente com probabilidade uniforme.

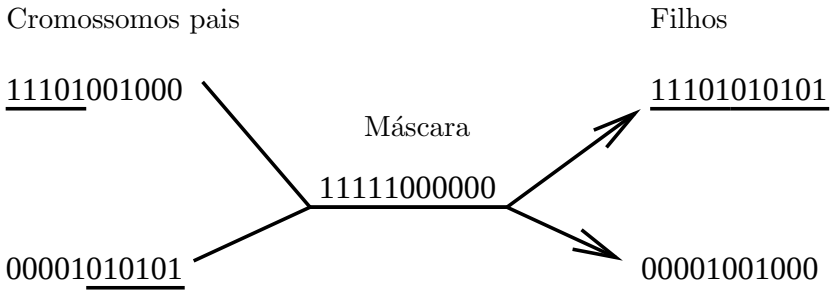


Figura 4.1: Operador *cross-over* de um ponto.

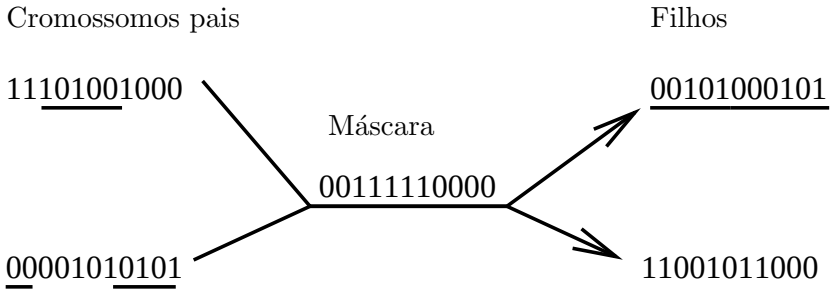


Figura 4.2: Operador *cross-over* de dois pontos.

4.2.7 Exemplo de Aplicação

O problema consiste em encontrar o valor máximo da função $f(x) = x^2$ onde x pode assumir valores do conjunto $A = \{0, \ldots, 31\}$. Para aplicação do algoritmo genético, os seguintes passos são sugeridos:

- Utilize 5 bits para representar uma solução candidata.
- Defina a função de aptidão $f_i(x) = f(x)$.
- Gere um conjunto de 4 soluções candidatas randomicamente para compor a população inicial. (Tamanho da população $p = 4$).

- Adote o operador *cross-over* de um ponto como método de reprodução. (Taxa de substituição por *cross-over* $r = 50\%$).
- Adote como mecanismo de mutação a troca de um bit de uma solução candidata. (Taxa de mutação $m = 5\%$).

Na Tabela 4.1 estão indicados os valores iniciais armazenados na memória do algoritmo genético. Com esta população inicial e os parâmetros acima sugeridos, o algoritmo genético encontra a solução ótima para o problema: $x^* = 11111$ sendo $f(x^*) = 961$.

Tabela 4.1: População inicial

Solução Candidata	Fitness	Probabilidade de Reprodução
01101	169	14,4%
11000	576	49,2%
01000	64	5,5%
10011	361	30,9%

4.2.8 Questões Práticas

O fenômeno de *crowding* ocorre quando um indivíduo com alta aptidão se reproduz rapidamente, gerando cópias ou filhos muito semelhantes que venham a constituir a maior parte da população. O aspecto negativo de *crowding* é a perda de diversidade, o que acarreta convergência prematura para ótimo local. O fenômeno de *crowding* é ilustrado na Figura 4.3.

Algumas estratégias para evitar *crowding* são:

- seleção de pares para reprodução através de torneios em vez de seleção probabilística;
- utilizar o valor de posição (*ranking*) no lugar da aptidão para calcular as probabilidades de selecionar os indivíduos; e
- o valor de aptidão de um indivíduo pode ser reduzido pela presença de outros indivíduos semelhantes.

4.2.9 *Schema Theorem*

É possível descrever matematicamente a evolução da população de um AG em função do tempo? O *Schema Theorem* de Holland (1975) [28] oferece uma caracterização.

Definição 4.1 *Um Schema é uma cadeia de 0s, 1s e *s (curinga) que representa uma família de indivíduos.*

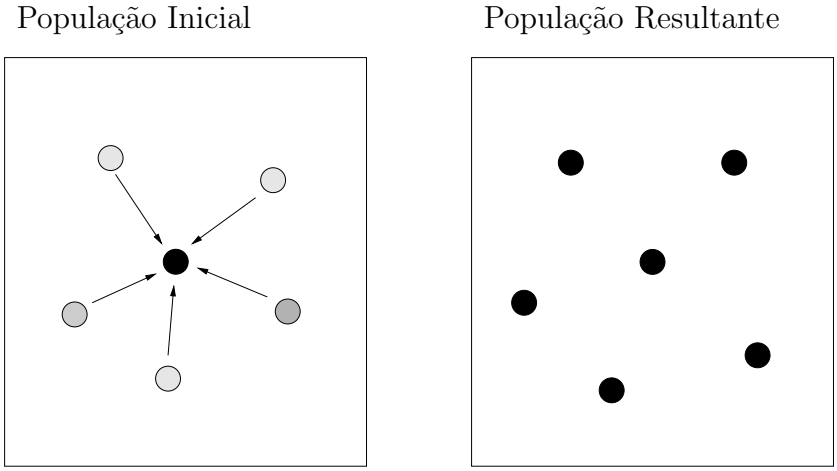


Figura 4.3: Fenômeno de *crowding*.

O *schema* "0*1" representa os indivíduos "001" e "011". *Schemas* podem ser utilizados para caracterizar a população de um algoritmo genético. Uma população de cadeias (vetores de bits 0s e 1s) pode ser vista em termos do conjunto de *schemas* que ela representa e o número de indivíduos associados com cada *schema*. Seja $m(s,t)$ o número de indivíduos no instante t (t-ésima população) que pertençam ao *schema* s. O *Schema Theorem* descreve o valor esperado[1] de $m(s,t+1)$ tomando como base: $m(s,t)$; as propriedades de s; as propriedades da população; e os parâmetros do AG (e.g., métodos de recombinação e taxa de mutação).

Teorema 4.1 *Considerando apenas seleção, tem-se*

$$E[m(s,t+1)] = \frac{u(s,t)}{f_m(t)} m(s,t)$$

onde: $m(s,t)$ *é o número de indivíduos em* P_t *(*t*-ésima população) que pertençam ao* schema s*;* $f_m(t)$ *é o valor médio da função de aptidão dos elementos de* P_t*; e* $u(s,t)$ *é o valor médio da função de aptidão dos elementos de* P_t *que pertençam ao* schema s*.*

Se vemos o AG como um processo virtual paralelo que i) executa uma busca através do espaço de possíveis *schemas* e, ao mesmo tempo, ii) executa uma busca através do espaço de indivíduos, então a equação do Teorema 4.1 indica que *schemas* de maior aptidão crescerão em influência com o passar das gerações.

Teorema 4.2 *Considerando o operador cross-over de um ponto e mutação, tem-se*

$$E[m(s,t+1)] \geqslant \frac{u(s,t)}{f_m(t)} m(s,t) \left[1 - \frac{p_c d(s)}{l-1}\right] [1-p_m]^{o(s)}$$

onde:

[1]Denotado por $E[\bullet]$ e mais rigorosamente conhecido por esperança matemática

- p_c *é a probabilidade do operador cross-over de um ponto ser aplicado a um indivíduo arbitrário;*
- p_m *é a probabilidade de que um bit arbitrário de um indivíduo arbitrário sofra mutação;*
- l *é o número de bits das cadeias;*
- $o(s)$ *é o número de bits definidos presentes no* schema s *(bits 0 ou 1); e*
- $d(s)$ *é a distância entre o bit definido mais à esquerda e o mais à direita no* schema s.

O Teorema 4.2 nos diz que os *schemas* de maior aptidão deverão aumentar sua influência, especialmente aqueles que possuem um pequeno número de bits definidos (contendo vários "*") e aqueles cujos bits definidos estão próximos uns dos outros.

4.3 *Simulated Annealing*

Simulated Annealing (SA) é um método de otimização bastante geral (como o Algoritmo Genético) que tem demonstrado excelente desempenho em problemas que apresentam um número muito grande de ótimos locais. Uma das aplicações mais bem-sucedidas de SA é o problema de localização de componentes eletrônicos em microprocessadores, no qual procura-se minimizar o comprimento total das conexões. A Figura 4.4 ilustra o problema de localização e empacotamento de circuitos eletrônicos.

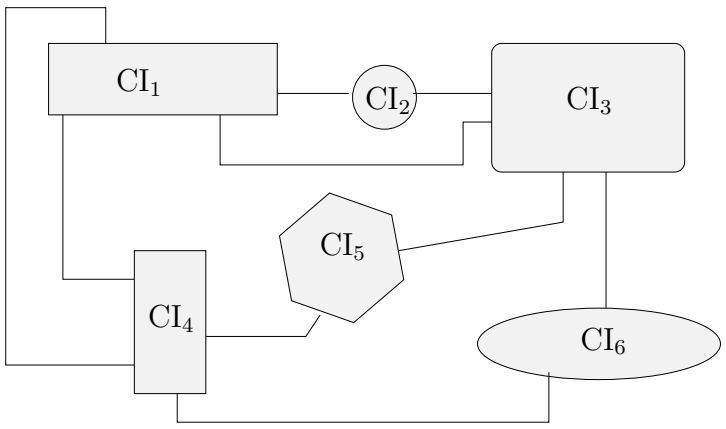


Figura 4.4: Empacotamento de componentes eletrônicos em circuitos integrados.

4.3.1 O Processo de *Annealing*

O processo de *annealing* se refere ao método por meio do qual líquidos (ou metais) se resfriam e cristalizam. Aquecemos a substância até uma temperatura muito alta, depois resfriamos lentamente. Na natureza, se o resfriamento é bastante lento, a substância

assume uma configuração de menor energia para a temperatura de equilíbrio. A Figura 4.5 ilustra o processo de formação de cristais por meio de *annealing*.

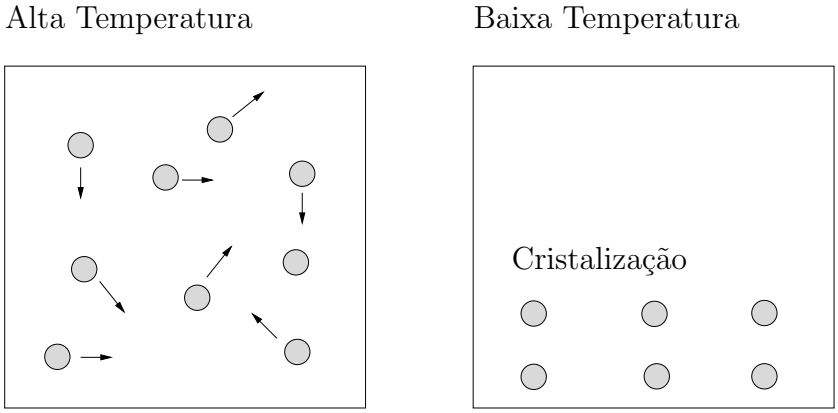


Figura 4.5: Ilustração do processo de *annealing*.

A Natureza pode ser vista como um processo de minimização do nível de energia, sendo este probabilístico. Mais precisamente, a probabilidade de um sistema físico se encontrar em uma configuração com nível de energia E é dada por:

$$Pr(E) \approx e^{-\frac{E}{\kappa T}} \tag{4.1}$$

onde κ é a constante de Boltzman e T é a temperatura. De acordo com a equação (4.1), para uma dada temperatura T, quanto maior o nível de energia, menor a probabilidade da configuração. Para um dado nível de energia E, a probabilidade da configuração diminui com a redução da temperatura. Mesmo em baixa temperatura é possível que o sistema esteja em um nível elevado de energia, embora a probabilidade seja extremamente pequena. Em outros palavras, o sistema aumenta o nível de energia algumas vezes, outras vezes ele diminui, mas a probabilidade de aumentar diminui exponencialmente com a redução da temperatura. Este comportamento oscilatório e decrescente com a redução da temperatura é ilustrado na Figura 4.6.

4.3.2 O Algoritmo de Metropolis

Em 1953 Metropolis et al. [36] incorporaram as idéias apresentadas acima em computações numéricas. Mais tarde, Kirkpatrick et al. [32, 31] estabeleceram a relação entre encontrar uma configuração de baixa energia e o problema de otimização, dando origem a um novo método de otimização inspirado em fenômenos físicos.

Dado um conjunto de configurações para um sistema, a probabilidade do sistema assumir a configuração E_2 a partir da configuração E_1 é dada por:

$$Pr = \begin{cases} e^{-(E_2-E_1)/kT} & \text{se } E_2 > E_1 \\ 1 & \text{se } E_2 < E_1 \end{cases}$$

A Figura 4.7 ilustra a queda de probabilidade de transição para estados de maior energia com a redução de temperatura. Para empregarmos o algoritmo de Metropolis em outros sistemas, não necessariamente termodinâmicos, precisamos de:

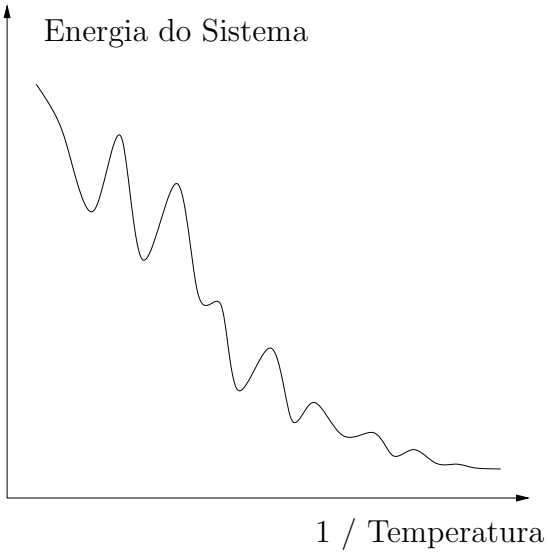


Figura 4.6: Comportamento típico do nível de energia conforme processo de *annealing*.

- uma descrição das possíveis configurações do sistema;
- um gerador randômico de perturbações do sistema; essas perturbações definem as "opções" de configuração;
- uma função objetivo (análoga à energia) cuja minimização desejamos executar; e
- um parâmetro de controle T (análogo à temperatura) e um procedimento de *annealing* que descreve a maneira pela qual T decresce.

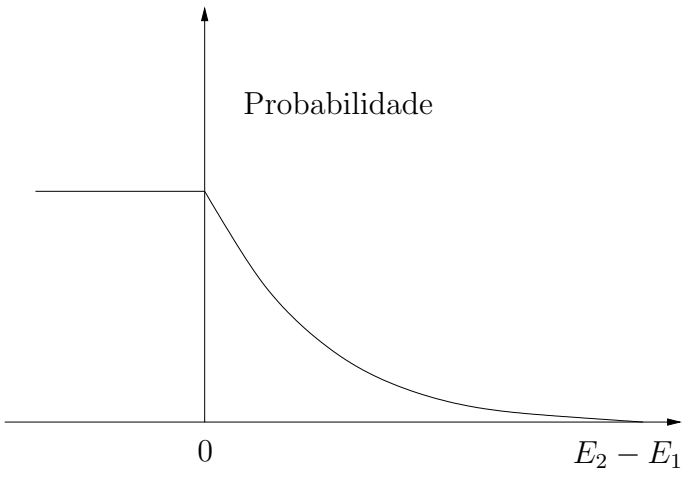


Figura 4.7: Probabilidade de transição com a queda de temperatura.

4.3.3 Exemplo: O Problema do Caixeiro Viajante

Em uma instância do Problema do Caixeiro Viajante (PCV) são dados N cidades e uma matriz de distâncias M. Desejamos encontrar uma rota que, partindo da cidade 1, visite as demais cidades precisamente uma vez e retorne à cidade 1, tal que o caminho percorrido seja o menor possível. O PCV pertence à família dos problemas NP-Difíceis [21, 12], o que significa dizer que não se conhece um algoritmo capaz de encontrar uma rota ótima em tempo polinomial qualquer que seja a instância do problema.

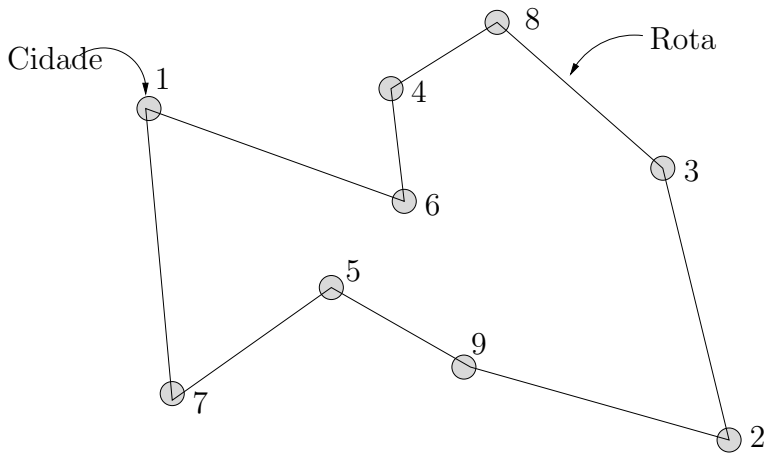


Figura 4.8: Exemplo de rota para uma instância particular.

Abaixo seguem os passos para se aplicar o algoritmo SA:

- **Configuração:** utilize um vetor x com N posições, $x \in \{1, 2, \ldots, N\}^N$, para armazenar uma permutação das cidades.
- **Perturbação:** uma subseqüência de cidades é removida da rota e inserida em outra posição. A Figura 4.9 ilustra dois operadores de perturbação.
- **Função Objetivo:** a função objetivo é definida como a equação:

$$E = \sum_{n=1}^{N-1} \|(x_n, y_n) - (x_{n+1}, y_{n+1})\| + \|(x_N, y_N) - (x_1, y_1)\|$$

- **Procedimento de Annealing:** abaixo segue os pseudo-código do procedimento de *annealing*.

Procedimento de *Annealing*

- Gere aleatoriamente um conjunto S de configurações
- Calcule a variação de energia ΔE máxima para as configurações de S:
  $\Delta E = \max\{|E[s_i] - E[s_j]| : s_i, s_j \in S\}$
- Defina T_0 (temperatura inicial) como algumas dezenas de vezes

o valor de ΔE

- Defina T_{min} como a temperatura mínima
- Seja s um elemento qualquer de S
- Seja $s_{best} = s$ a melhor solução encontrada
- Faça $k = 0$
- Equanto $T_k \geqslant T_{min}$ faça

Repita para $n = 1$ até $100N$
Obtenha uma solução alternativa s' perturbando s
Se $E[s'] < E[s]$ então
$s \leftarrow s'$
Se $E[s] < E[s_{best}]$ então
$s_{best} \leftarrow s$
Fim-se
Caso contrário
Faça $s \leftarrow s'$ com probabiblidade dada por $e^{-\frac{E[s']-E[s]}{\kappa T}}$
Fim-se
Fim-repita
$T_{k+1} \leftarrow 0.98T_k$
$k \leftarrow k + 1$

- Fim-Enquanto

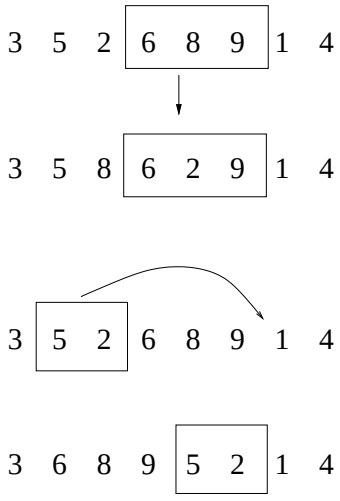


Figura 4.9: Ilustração dos operadores de perturbação

4.4 Referências

Os livros de Davis [15] e de Goldberg [24] trazem uma discussão ampla da filosofia por trás dos Algoritmos Genéticos, apresentam elementos teóricos e aplicações. Tais

referências são indicadas para um aprofundamento na teoria e aplicações de Algoritmos Genéticos. Recomendamos ainda dois *surveys* sobre Algoritmos Evolucionários, um voltado para a solução de problemas não-lineares inteiros mistos [13] e outro voltado para problemas encontrados nas áreas de sistemas e teoria de controle [19]. Para um estudo teórico da propriedade de convergência global do método *simulated annealing*, o leitor pode consultar o livro de Aart e Korst [1].

4.5 Exercícios

EX 4.1 (O Problema de Bi-Partição Máxima em Grafos): Seja $G = (V, E)$ um grafo não direcionado cujas arestas possuem pesos não-negativos. Uma aresta $(u, v) \in E$ possui peso $w(u, v)$. Para dois conjuntos A e B de vértices, definimos $w(A, B)$ o peso total das arestas que têm um extremo em A e o outro, em B. Mais precisamente:

$$w(A, B) = \sum_{(u,v)\in E, u\in A, v\in B} w(u, v)$$

O problema é encontrar $S \subseteq V$ tal que $w(S, V - S)$ seja máximo.

Os dados de uma instância do problema da bi-partição máxima são fornecidos em arquivos tipo texto, cujo formato consiste do número de vértices, do número de arestas e de uma lista de arestas e pesos.

Exemplo de instância:
5 6
1 3 0.3
2 4 0.5
2 5 1.2
1 5 0.8
3 4 0.5
2 3 3.0

Tarefas

i. implemente um Algoritmo Genético para resolver o problema de bi-partição máxima e teste o algoritmo em alguns problemas;

ii. implemente e teste um algoritmo Simulated Annealing para o problema de bi-partição máxima; e

iii. compare os algoritmos e os resultados obtidos.

EX 4.2: O problema da bi-partição máxima é NP-Difícil? Um problema é NP-Difícil se qualquer problema de decisão NP-Completo pode ser reduzido em tempo polinomial ao problema em consideração.

EX 4.3: O problema da bi-partição mínima é formulado como:

$$\begin{array}{ll} \text{Minimize} & w(S, V-S) \\ S \subseteq V & \\ \text{Sujeito a:} & \\ & S \neq \emptyset \end{array}$$

Como que este problema poderia ser resolvido? O problema da bi-partição mínima é NP-Difícil?

EX 4.4: Suponha que os pesos das arestas do grafo $G = (V, E)$ são variáveis aleatórias independentes: o peso da aresta (u, v) é uma variável aleatória $w(u, v)$ com distribuição, média e variança conhecidos que independem dos pesos das demais arestas. Em tal situação, como que se poderia resolver os problemas da bi-partição máxima e mínima se o objetivo é maximizar ou minimizar o valor esperado $E[w(S, V-S)]$ respectivamente?

Capítulo 5

Treinamento de Redes Neurais: Um Problema de Otimização

Redes Neurais Artificiais (RNs) constituem métodos "robustos" de aproximação de funções de valores vetoriais, reais ou discretos. Em outras palavras, as redes neurais toleram erros nos pares de treinamento. Uma rede neural pode ser vista como um aproximador universal ou, mais precisamente, como um interpolador universal uma vez que elas não são capazes de inferir informações que estão fora do conjunto de treinamento. Dado um conjunto qualquer de treinamento podemos, em princípio, construir uma rede neural que produz os mesmos valores da função para os pares entrada-saída de treinamento. O algoritmo de treinamento de propagação reversa (*back-propagation*) nada mais é do que uma implementação especializada do algoritmo de descenso, sendo objeto de estudo neste capítulo.

Como o próprio nome sugere, redes neurais são inspiradas em sistemas biológicos de aprendizagem que tipicamente consistem de redes complexas de unidades elementares, conforme a Figura 5.1. Na figura, a primeira camada recebe os sinais de entrada, que são processados e transformados em novos sinais a serem transmitidos às unidades da camada intermediária, que por sua vez processa os sinais de entrada e os envia à camada de saída.

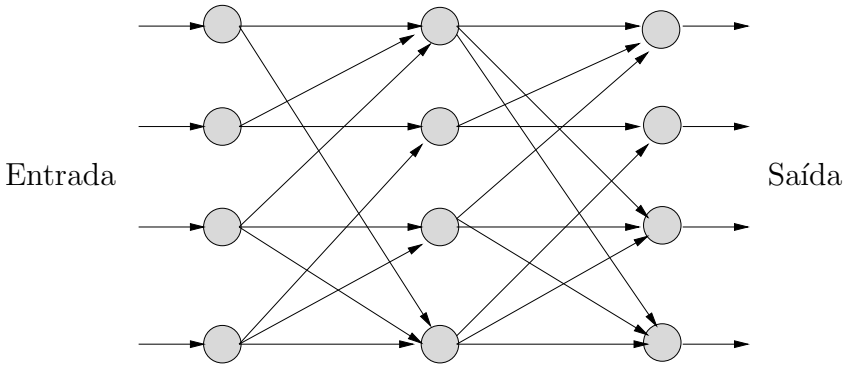


Figura 5.1: Exemplo de topologia de redes neurais.

É interessante considerarmos que o cérebro humano possui aproximadamente 10^{11}

neurônios (100 bilhões de neurônios), cada um conectado com cerca de 10^4 neurônios. O tempo de resposta de um neurônio é aproximadamente 10^{-3} segundos, muito mais lento do que um computador, cujas portas lógicas apresentam um tempo de resposta da ordem de 10^{-10} segundos. O cérebro humano reconhece uma figura em cerca de 10^{-1} segundos. Dado o tempo relativamente lento de resposta do neurônio, em comparação com a velocidade de processamento de informação, podemos concluir que o sistema de aprendizagem biológico é extremamente paralelo e distribuído, todavia não possui um número muito grande de camadas.

Existem, entretanto, incompatibilidades entre as redes neurais artificiais e as unidades do sistema biológico de aprendizagem. A saída de uma unidade artificial é um valor constante, enquanto que as unidades biológicas emitem uma série complexa de sinais, consistindo em pulsos com várias freqüências.

5.1 Elementos Básicos das Redes Neurais

O elemento básico de uma rede neural é a unidade de processamento. Na Figura 5.2 é ilustrada a estrutura geral de uma unidade de processamento neural, a qual consiste em um vetor com os sinais de entrada $x = (x_1, \ldots, x_n)$, uma unidade de combinação linear dos sinais de entrada cujos pesos são dados pelo vetor $w = (w_1, \ldots, w_n)$, e uma função de ativação $f(s)$ onde $s = \sum_{i=1}^{n} w_i x_i$ que produz os sinais de saída tal que $y_i = f(s)$ para $i = 1, \ldots, m$. Unidades de processamento são organizadas em camadas e conectadas por arcos (sinapses) que simbolizam os sinais de comunicação de uma camada (ou sinais de entradas) para outra camada. Um exemplo de rede neural é dado na Figura 5.3.

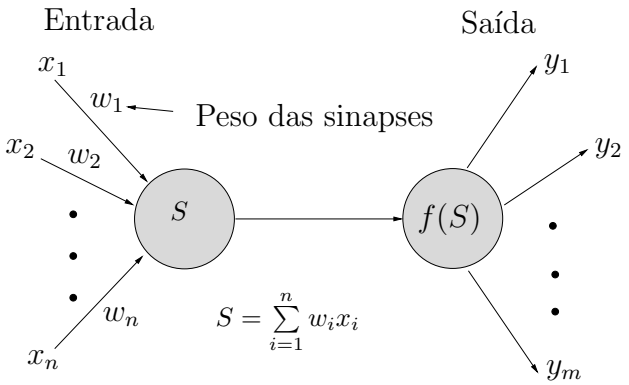


Figura 5.2: Unidade de processamento neural.

5.1.1 O Problema de Treinamento

Dados: uma rede neural (a sua topologia), ou seja, um grafo $G = (V, E)$; exemplos de treinamento, ou seja, uma lista de pares de entrada e saída desejados $D = \{(x^1, y^1), \ldots, (x^K, y^K)\}$.

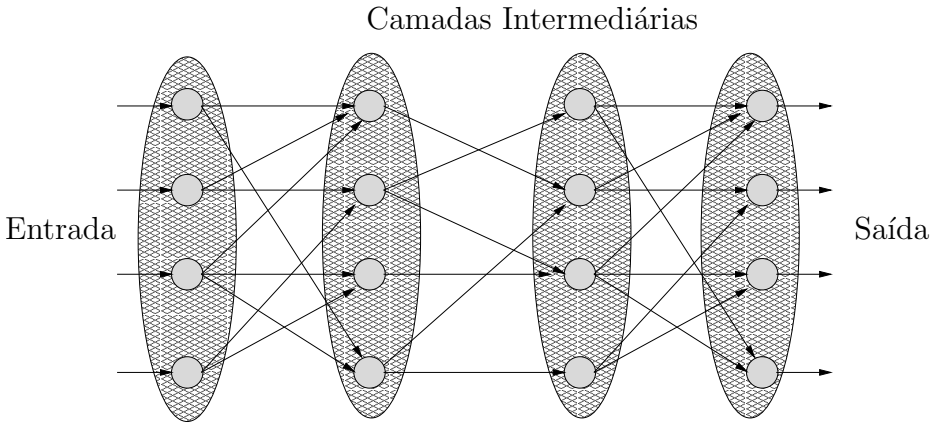


Figura 5.3: Rede neural com camadas intermediárias.

O problema é encontrar pesos para as sinapses tal que o erro de estimação das saídas seja o menor possível (*least squares problem*). Seja $f(w, x) : \mathbb{R}^n \to \mathbb{R}^m$ a função representada pela rede neural com pesos dados por um vetor $w \in \mathbb{R}^r$, onde r é o número de conexõe sinápticas da rede. Dado x, a rede responde com um valor $y = f(x)$. Em programação matemática, o problema de treinamento da rede neural pode ser escrito como:

$$\begin{aligned} &\text{Minimize} \quad \tfrac{1}{2} \sum_{k=1}^{K} \|f(w, x^k) - y^k\|^2 \\ &w \in \mathbb{R}^r \end{aligned}$$

5.1.2 ALVINN: Um Exemplo de Aplicação

Um exemplo de aplicação de redes neurais é o sistema ALVINN [40] desenvolvido na Universidade de Carnegie Mellon, que utilizou com sucesso o modelo de redes neurais para dirigir de forma autônoma um veículo por cerca de 120 Km a uma velocidade de até 100 Km/h. Mais recentemente, este mesmo sistema foi aprimorado e desempenhou com sucesso a tarefa mais audaciosa de dirigir o mesmo veículo, de uma forma semi-autônoma, da cidade de Pittsburgh na Pennsylvania até a cidade de Berkeley na California. Uma ilustração simplificada da rede neural do sistema ALVINN é mostrada na Figura 5.4. Cada pixel da imagem é primeiramente transformado em tonalidade de cinza e depois enviado à respectiva unidade neural, que por sua vez processa o sinal recebido e retransmite às outras unidades da camada intermediária. A camada neural de saída conta com várias unidades que indicam o grau de giro do volante, onde a unidade mais à esquerda indica giro máximo à esquerda enquanto que a unidade mais à direita indica giro oposto. A unidade cujo valor de saída é mais intenso terá seu comando executado pelo sistema robótico.

5.1.3 Problemas Apropriados para Redes Neurais

Em geral, as redes neurais são recomendadas para problemas de aproximação complexos tipicamente de natureza não-linear e sujeitos a ruídos, tais como dados e sinais

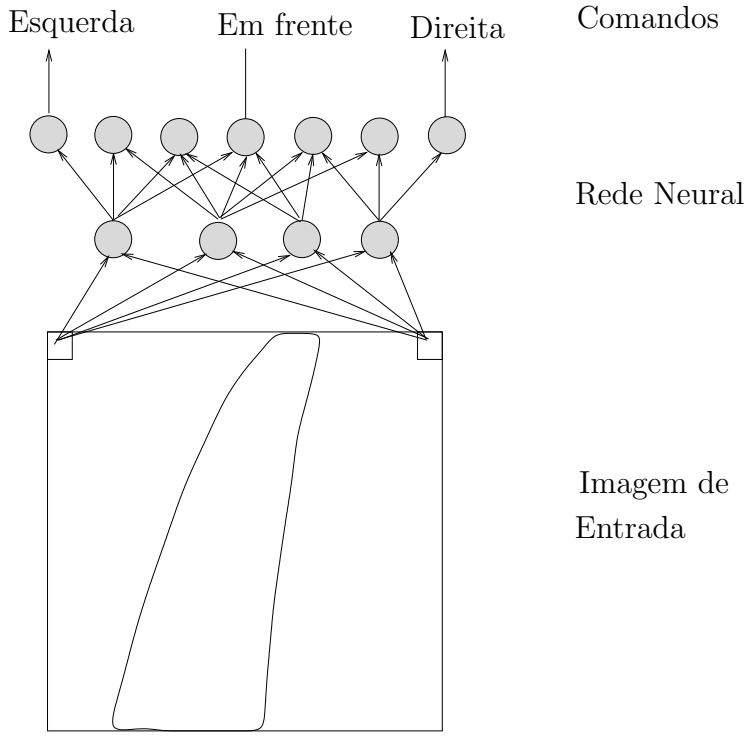


Figura 5.4: Rede neural do sistema ALVINN.

provenientes de câmeras e microfones. Mais especificamente, RNs em combinação com o algoritmo de propagação reversa (*back-propagation*) são adequadas para problemas com as seguintes características:

- instâncias que são representadas por pares contendo muitos atributos (múltiplas entradas e múltiplas saídas), ou seja, as dimensões dos vetores x e y;
- o atributo de saída tenha um valor discreto, real ou um vetor de valores; no ALVINN, por exemplo, a saída é um vetor com cerca de 30 saídas, cada uma representando a probabilidade de um comando.
- treinamento pode ser feito *off-line*;
- a avaliação da função deve ser muito rápida, para uso em aplicações de tempo-real;
- a habilidade de se entender o significado da aproximação não é necessária, o que pode ser um obstáculo para o uso de redes neurais em aplicações de diagnóstico médico, onde é imprescindível que o sistema justifique uma decisão e indique a seqüência de inferência que leva à conclusão; e
- os exemplos de treinamento podem conter erros.

5.2 Perceptron: A Primeira Unidade Neural

Marvin Minsky e Dean Edmonds construiram o primeiro computador neural em 1951. Ironicamente, Minsky mais tarde provou teoremas que contribuíram para o fim das pesquisas em redes neurais em meados de 1970 [37]. Embora a Ciência da Computação tenha desprezado o campo das redes neurais após a publicação do livro Perceptrons, por Minsky e Papert, trabalhos de pesquisa continuaram em outros domínios, especialmente na Física. A motivação principal veio nos anos 80 quando pelo menos 4 grupos reinventaram o algoritmo de propagação reversa (*back-propagation algorithm*), inicialmente proposto por Bryson e Ho em 1969 [10]. Os algoritmos foram aplicados a vários problemas de aprendizagem em Ciência da Computação e Psicologia.

A estrutura básica da unidade perceptron é ilustrada na Figura 5.5. Em essência, a unidade recebe o sinais de entrada $x_1, \ldots, x_n$, pois assumimos que o sinal x_0 é sempre 1, e os combina linearmente de acordo com os pesos sinápticos $w_0, \ldots, w_n$, obtendo o sinal $w^T x$, onde $x = (x_0, \ldots, x_n)$ e $w = (w_0, \ldots, w_n)$. O sinal de saída enviado pelos canais $y_1, \ldots, y_m$ são idênticos, assumindo o valor 1 se o sinal de $w^T x$ é positivo e -1 caso contrário. A função de saída se comporta, portanto, como um *threshold*: se $(w_1, \ldots, w_n)^T (x_1, \ldots, x_n) > w_0$ onde w_0 contém o valor do *threshold* então o sinal de saída assume o valor máximo, 1, de outra forma o sinal assume o valor mínimo, -1.

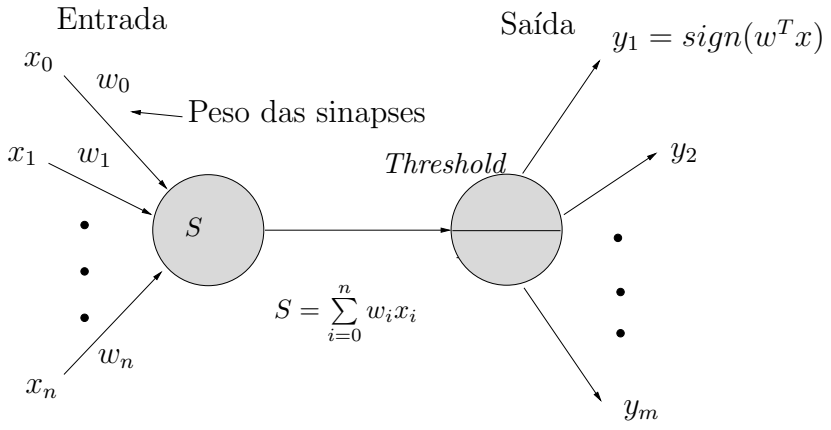


Figura 5.5: Perceptron.

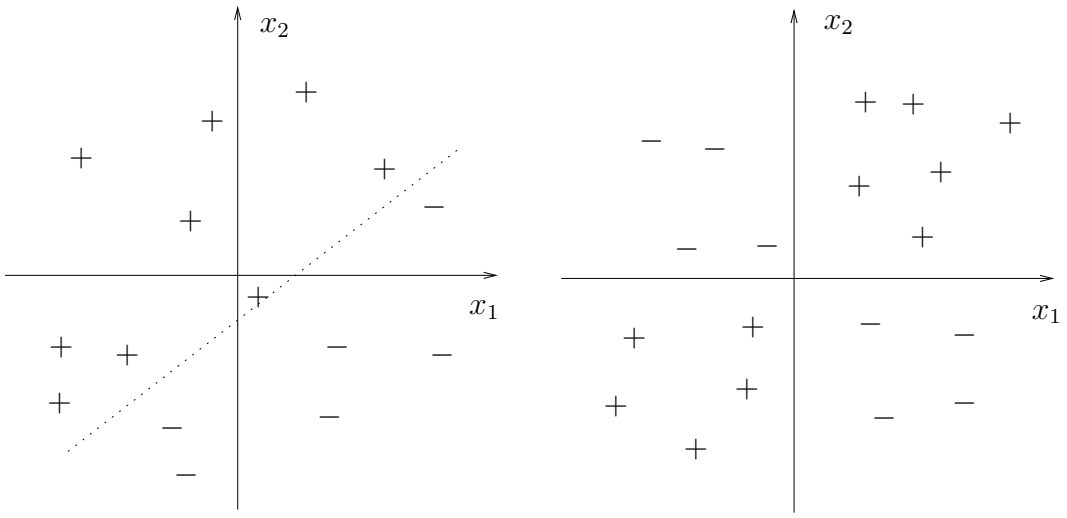
5.2.1 Treinando um Perceptron

Sendo dado um conjunto de pontos de treinamento $D = \{(x^1, t^1), \ldots, (x^M, t^M)\}$, considere o problema de treinamento de um perceptron, que consiste em minimizar o erro de predição:

$$P: \quad \underset{w \in \mathbb{R}^n}{\text{Minimize}} \quad \sum_{k=1}^{M} \left(sign(w^T x^k) - t^k \right)^2$$

Quais são os possíveis obstáculos a serem encontrados na resolução de P? Um obstáculo fundamental é o fato de que a função $sign(x)$ não é diferenciável, o que inviabiliza o emprego de algoritmos eficientes baseados no gradiente da função. Outra limitação

fundamental advém do fato que o perceptron só consegue classificar corretamente, com erro nulo, aqueles conjuntos de dados que são linearmente separáveis. Note que a região $H = \{x \in \mathbb{R}^n : sign(w^T x) \leqslant 0\}$ forma um hiper-espaço, sendo a região complementar $\bar{H}$ dada por $\bar{H} = \{x \in \mathbb{R}^n : sign(w^T x) > 0\}$. Em outras palavras, o perceptron define um subespaço multidimensional, classificando como positivo os exemplos que estão de um lado e como negativo os demais. Esta limitação é ilustrada na Figura 5.6. O perceptron pode ser usado para representar as funções lógicas *and*, *or*, *nor* e *nand*. Por si só, entretanto, o perceptron não consegue implementar a porta lógica *xor*. Todavia, podemos aumentar o poder de representação do peceptron se utilizarmos uma rede. Qualquer função Booleana dada na forma normal disjuntiva (i.e., uma disjunção de cláusulas) pode ser representada por uma rede de perceptrons.



Conjunto Linearmente Separável *Conjunto Não Separável Linearmente*

Figura 5.6: Conjuntos separáveis e não separáveis linearmente.

O procedimento de treinamento de um perceptron é o elemento básico para desenvolver o procedimento de treinamento de uma rede. Assim apresentamos abaixo um algoritmo de treinamento de um perceptron.

Algoritmo de Treinamento de um Perceptron

O vetor com pesos sinápticos $w = (w_0, \ldots, w_n)$
    é inicializado randomicamente $(-0,5 < w_i < 0,5)$
$\alpha \leftarrow$ taxa de aprendizagem (e.g., $\alpha = 0,01$)
$m \leftarrow 1$ (temos M exemplos de treinamento)
Repita
    Para $i = 0, \ldots, n$
        $\Delta w_i = \alpha[t^m - sign(w^T x^m)]x_i^m$
        $w_i = w_i + \Delta w_i$
    Fim-Para
    $m \leftarrow mod(m, M) + 1$

Fim-Repita

Uma propriedade interessante do algoritmo acima é que ele converge sempre que o conjunto de treinamento for linearmente separável [37].

5.3 Regra Delta

Um dos obstáculos ao uso do perceptron é a possibilidade de falha quando os exemplos de treinamento não são linearmente separáveis. A regra delta supera esta dificuldade, propiciando a concepção de um algoritmo que converge para a melhor aproximação linear da relação entre entrada e saída (a função entrada-saída desconhecida). Em síntese, a regra delta consiste em substituir a saída $o(x) = sign(w^T x)$ pela saída $o(x) = w^T x$, i.e., simplesmente eliminamos a função sinal deixando a função de saída o uma função linear e diferenciável. A unidade delta é ilustrada na Figura 5.7.

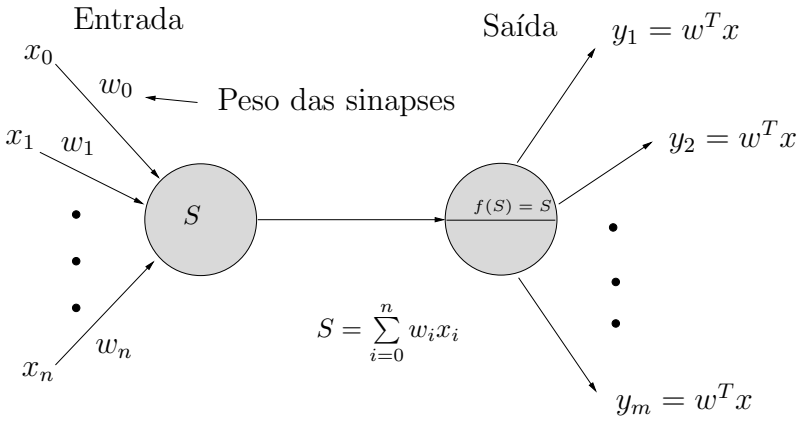


Figura 5.7: Unidade delta.

5.3.1 Treinando uma Unidade Delta

Dado um conjunto fixo de exemplos $D = (x^1, t^1), \ldots, (x^M, t^M)$, onde $x^k \in \mathbb{R}^{n+1}$ e $t^k \in \mathbb{R}$, $k = 1, \ldots, M$, o problema de treinamento da unidade delta consiste em encontrar o vetor $w \in \mathbb{R}^{n+1}$ que minimize o erro de predição. (Note que assumimos que $x_0^k = 1$.) Em programação matemática, o problema é expresso como:

$$P: \quad \begin{array}{ll} \text{Minimize} & E(w) = \frac{1}{2} \sum_{k=1}^{M} \left(w^T x^k - t^k\right)^2 \\ w \in \mathbb{R}^{n+1} & \end{array}$$

Portanto, o problema de minimizar a função $E(w)$ está dentro do domínio da otmização diferenciável sem restrições. Podemos, por exemplo, aplicar o algoritmo de descenso ou algoritmo de Newton. Qualquer que seja a abordagem, será necessário calcular

o gradiente de E. Seja $\nabla E = [\partial E/\partial w_0, \partial E/\partial w_1, \ldots, \partial E/\partial w_n]$ o gradiente de E. Podemos calcular a derivada parcial de E com respeito a w_j como segue:

$$\begin{aligned}\frac{\partial E}{\partial w_j} &= \frac{\partial}{\partial w_j}\left\{\frac{1}{2}\sum_{k=1}^{M}\left(w^T x^k - t^k\right)^2\right\} \\ &= \sum_{k=1}^{M}\left(w^T x^k - t^k\right)\frac{\partial}{\partial w_j}\left(w^T x^k - t^k\right) \\ &= \sum_{k=1}^{M}\left(w^T x^k - t^k\right)x_j^k\end{aligned}$$

Note que ∇E pode ser convenientemente calculado em termos dos valores das saídas para as diferentes entradas, $w^T x^k - t^k$, e o próprio valor da k-ésima entrada x^k. Se utilizarmos um algoritmo de descenso para resolver o problema, então dada uma solução candidata w^l, o algoritmo de descenso busca a solução alternativa $w^{l+1} = w^l - \alpha^l \nabla E(w^l)$, sendo α^l a solução (aproximada) do problema:

$$\begin{array}{ll}\text{Minimize} & f_l(\alpha) = E\left[w^l - \alpha^l \nabla E(w^l)\right] \\ \alpha \in \mathbb{R} & \\ \text{Sujeito a :} & \alpha \geqslant 0\end{array}$$

A capacidade representacional de uma rede com múltiplas camadas de unidades lineares é superior à capacidade de uma unidade linear? Não, pois uma composição linear de funções lineares continua sendo uma função linear. Assim, uma rede neural multi-camada de unidades delta não é mais representativa do que o espaço das funções lineares.

5.4 A Unidade *Sigmoid*

Objetivando contornar as limitações da unidade delta e, ao mesmo tempo, incorporar o poder representacional de uma rede de perceptrons, foi proposto a aproximação da função *threshold* com uma função diferenciável. Uma função com tais propriedades é a função *sigmoid* $\sigma(x)$, cujo comportamento é ilustrado na Figura 5.8.

Dados i) um grafo $G = (V, A)$ representando uma rede com múltiplas camadas, cujas unidades de processamento neural são do tipo *sigmoid*, e ii) uma lista com exemplos de treinamento $(x^1, t^1), \ldots, (x^M, t^M)$, onde $x^k \in \mathbb{R}^{n+1}$ e $t^k \in \mathbb{R}^m$, podemos utilizar o algoritmo de propagação reversa para treinar a rede. (Lembramos que $x^k = [x_0^k, x_1^k, \ldots, x_n^k]$, $w^k = [w_0^k, w_1^k, \ldots, w_n^k]$ e $x_0^k = 1$.) Este algoritmo consiste em um algoritmo de descenso que objetiva minimizar o erro entre as saídas da rede e os valores desejados. Em programação matemática, este problema pode ser especificado como:

$$\begin{array}{lll}P: & \text{Minimize} & E(w) = \frac{1}{2}\sum_{k=1}^{M}\sum_{j \in Saida}\left\|o_j(x^k) - t^k\right\|^2 \\ & w \in \mathbb{R}^r & \end{array}$$

onde w é o vetor com os pesos sinápticos de todos os arcos de G e $o_j(x)$ é a saída da j-ésima unidade da camada de saída da rede neural.

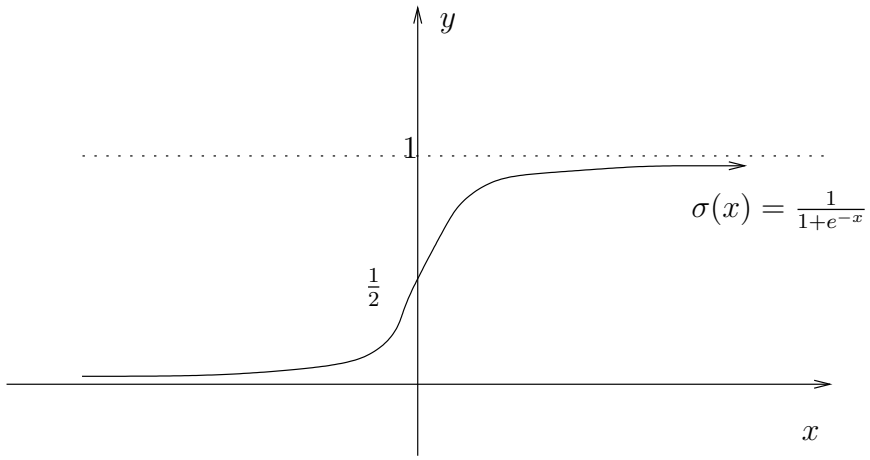


Figura 5.8: Função *sigmoid*.

Redes multi-camada com unidades *sigmoid* podem representar funções não lineares de elevada complexidade. Elas podem ser utilizadas em reconhecimento de voz, identificação e modelagem de processos não-lineares, e predição de grandezas econômicas. Na Figura 5.9 é ilustrado um sistema neural para reconhecimento de palavras que toma como entrada dois sinais com a intensidade de duas freqüências.

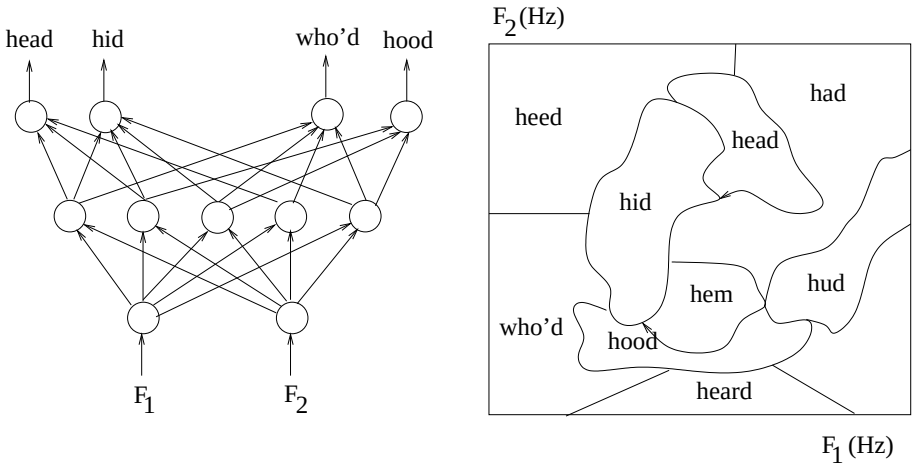


Figura 5.9: Sistema neural para reconhecimento de palavras

5.5 Referências

O presente capítulo é uma síntese da parte de Redes Neurais do livro de Mitchell [38]. Para uma exposição rigorosa do algoritmo *back-propagation* recomendamos o livro de Bertsekas [9].

Uma abordagem unificada de Redes Neurais com enfoque em modelagem e sistemas de controle aparece no artigo de Hagan e Demuth [26]. Este artigo propõe uma modelagem genérica de redes tipo perceptron multicamadas, introduzindo conceitos fundamentais, discutindo o emprego de redes Neurais como aproximadores universais e descrevendo um algoritmo de treinamento tipo *back-propagation*. Por fim os autores fazem uma breve apresentação dos modelos de controladores baseados em Redes Neurais. Para um aprofundamento maior das abordagens de controle, recomendamos os seguintes artigos de revisão: [29], [6], [2].

5.6 Exercícios

EX 5.1: Implemente com um perceptron as funções lógicas *AND*, *NAND*, *OR* e *NOR*.

EX 5.2: Considere o grafo $G = (V, E)$ da Figura 5.10, o qual representa uma rede neural com três camadas: a camada de entrada com quatro unidades (vértices do grafo); a camada intermediária com duas unidades; e a camada de saída com quatro unidades. Seja o_j o valor da saída de uma unidade j. As unidades de entrada simplesmente copiam os valores de entrada para suas saídas, ou seja, $(o_1 = x_1, \ldots, o_4 = x_4)$. As saídas da rede neural correspondem às saídas das unidades da terceira camada $(o_7 = y_1, \ldots, o_{10} = y_4)$.

Seja j uma unidade intermediária ou de saída. Esta unidade j calcula sua saída o_j como segue:

$$net_j = \sum_{(i,j)\in E} w_{ij} o_i$$

$$o_j = \sigma(net_j) = \frac{1}{1 + e^{-net_j}}$$

O conjunto de entradas e saídas desejadas é dado na Tabela 5.1. Dados os pesos dos arcos de G $(w_{15}, w_{16}, w_{25}, w_{26}, \ldots, w_{67}, w_{68}, w_{69}, w_{6,10})$, a rede neural pode ser vista como uma função $f_w(x) : \mathbb{R}^4 \to \mathbb{R}^4$ que, dado o valor de x, retorna o valor estimado de y.

Tabela 5.1: Amostras para treinamento da rede neural

	Amostra	Saída
t	$x^t = (x_1^t, x_2^t, x_3^t, x_4^t)$	$(y_1^t, y_2^t, y_3^t, y_4^t)$
1	(0,0,0,1)	(0,0,0,1)
2	(0,0,1,0)	(0,0,1,0)
3	(0,1,0,0)	(0,1,0,0)
4	(1,0,0,0)	(1,0,0,0)

Tarefas

i. Formule o problema de treinar esta rede neural em programação matemática. (Sugestão: defina restrições para calcular net_j^t e o_j^t, para cada unidade j e exemplo de treinamento t, como uma função dos pesos w's e das variáveis net_j^t e o_j^t que antecedem a unidade em questão; formule uma função objetivo para minimizar o erro entre as saídas da rede neural e os valores desejados, i.e, o erro entre o_{6+k}^t e o resultado esperado y_k^t, para $k = 1, \ldots, 4$.)

ii. Se você utilizou restrições, reescreva o problema em programação matemática sem utilizar restrições.

iii. Aplique o algoritmo de descenso para encontrar os valores dos pesos sinápticos w que minimiza o erro.

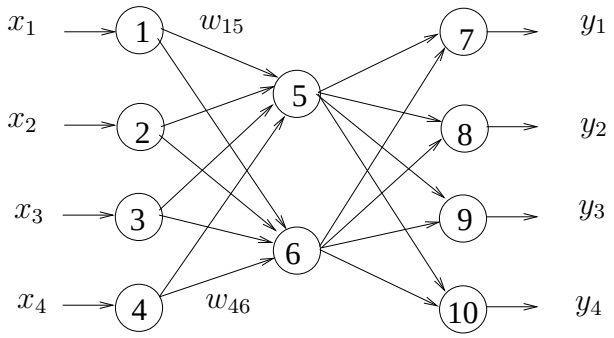


Figura 5.10: Topologia da rede neural.

Capítulo 6

Programação Linear

A programação linear teve sua incepção nos anos 30 e foi impulsionada nos anos 40 com o desenvolvimento do algoritmo Simplex por George Dantzig. Aplicações em engenharia, ciência da computação, economia e matemática são inúmeras. Alguns pesquisadores consideram que os algoritmos de programação linear são os mais empregados na ciência da computação. Existem algoritmos que são versões especializadas do algoritmo Simplex como, por exemplo, o algoritmo Simplex para problemas de fluxo em redes. Em 1979, Khachiyan propôs o primeiro algoritmo de tempo polinomial para programação linear. Este algoritmo, conhecido por elipsóide, não teve sucesso prático em virtude do tamanho das matrizes geradas, pela necessidade de precisão numérica acentuada e em decorrência de instabilidade numérica. O algoritmo elipsóide tem conseqüências relevantes na programação inteira e na solução de problemas lineares com um número exponencial de restrições[1]. Mais recentemente, em 1984, Karmakar [30] desenvolveu um algoritmo de ponto-interior com tempo polinomial e potencial para resolver instância práticas. Surge então uma área intensa de pesquisa que se alastra até os dias atuais: algoritmos de ponto-interior para programação linear, programação quadrática e outros problemas. Apesar do tempo polinomial dos algoritmos de ponto-interior, o algoritmo Simplex continua sendo muito empregado, em parte porque ele ainda é mais eficiente em uma gama considerável de problemas e e também em função de suas aplicações em programação inteira.

Os assuntos tratados neste capítulo são uma compilação de tópicos dos textos de Chvatal [11] e Vanderbei [42]. A Seção 6.1 ilustra uma aplicação de programação linear e o conceito de problema dual. A Seção 6.2 apresenta a formulação geral do problema de programação linear, enquanto a Seção 6.3 aborda o algoritmo Simplex por meio de dicionários. A Seção 6.4 trata da teoria da dualidade em programação linear, apresentando conceitos gerais, os teoremas fraco e forte da dualidade, e o teorema da folga complementar. Por fim, a Seção 6.5 formaliza o algoritmo Simplex em notação compacta matricial.

[1]Se um problema de programação linear tem um número exponencial de restrições, mas existe um algoritmo de tempo polinomial capaz de detectar uma restrição não satisfeita por uma solução candidata, então o algoritmo elipsóide leva a um algoritmo de tempo polinomial para resolver o problema em questão.

6.1 Problema Exemplo: Gerenciamento de Uma Unidade de Produção

Uma unidade de produção pode manufaturar uma variedade de produtos $N = \{1, \ldots, n\}$, sendo estes produtos montados a partir de um conjunto de matérias-primas $M = \{1, \ldots, m\}$. As decisões gerenciais são dinâmicas, evoluindo conforme as condições do mercado. Considere um certo momento onde: b_i é a quantidade de matéria-prima i disponível; ρ_i é o custo unitário da matéria-prima i; a manufatura de uma unidade de produto j requer a_{ij} unidades da matéria-prima i; e o produto j pode ser vendido ao preço σ_j por unidade. A seguir investigaremos dois problemas relacionados.

6.1.1 Gerente de Produção Otimista

O problema envolve a alocação da matéria-prima na manufatura de itens acabados. Se o gerente decide produzir x_j unidades do produto j, então o lucro associado com a venda de uma unidade de produto j é $c_j = \sigma_j - \sum_{i=1}^{m} \rho_i a_{ij}$. O problema de determinar o número de unidades a serem vendidas de forma a maximizar o lucro pode ser expresso em programação matemática como segue:

$$\begin{array}{lll} P_0: \text{ Maximize } & \sum_{j=1}^{n} c_j x_j & \\ \text{Sujeito a :} & & \\ & x_j \geqslant 0 & j = 1, \ldots, n \\ & \sum_{j=1}^{n} a_{ij} x_j \leqslant b_i & i = 1, \ldots, m \end{array} \tag{6.1}$$

Assim, a tarefa do gerente é determinar as quantidades de produção, $\{x_j\}_{j=1}^{n}$, que maximizem a função objetivo e satisfaçam as restrições de (6.1).

6.1.2 Gerente de Produção Pessimista

Nesta situação o gerente assume a posição de contador tendo como tarefa avaliar o preço de venda de cada matéria-prima. Em outras palavras, a unidade de manufatura não produzirá e passará a se comportar como um depósito. A unidade deve estar preparada para vender a matéria-prima. Assim seja w_i o valor unitário para vender a matéria-prima i, a ser definido pelo gerente. O valor total da venda de todo o estoque é $\sum_{i=1}^{m} b_i w_i$. O objetivo do gerente é encontrar o valor mais baixo para venda que não incorra perdas financeiras. Portanto, duas restrições são:

- $w_i \geqslant \rho_i, i = 1, \ldots, m$, o valor do item i deve ser maior ou igual ao valor de mercado; e
- $\sum_{i=1}^{m} a_{ij} w_j \geqslant \sigma_j, j = 1, \ldots, n$, pois de outra forma seria mais lucrativo utilizar a matéria-prima para gerar produto j e vendê-lo no mercado.

O problema a ser resolvido pelo gerente então fica:

$$\begin{aligned} P_1: \quad & \text{Minimize} \quad \sum_{i=1}^{m} b_i w_i \\ & \text{Sujeito a :} \\ & \qquad w_i \geqslant \rho_i \qquad\qquad i = 1, \ldots, m \\ & \qquad \sum_{i=1}^{m} a_{ij} w_i \geqslant \sigma_j \qquad j = 1, \ldots, n \end{aligned}$$

Agora, efetue a mudança de variável conforme segue:

$$w_i \geqslant \rho_i \quad \Leftrightarrow \quad w_i - y_i = \rho_i \text{ e } y_i \geqslant 0 \tag{6.2}$$

$$\Leftrightarrow \quad w_i = y_i + \rho_i \text{ e } y_i \geqslant 0 \tag{6.3}$$

Em termos das variáveis $\{y_i\}_{i=1}^{m}$, o problema se torna:

$$\begin{aligned} P_2: \quad & \text{Minimize} \quad \sum_{i=1}^{m} b_i y_i + \sum_{i=1}^{m} b_i \rho_i \\ & \text{Sujeito a :} \\ & \qquad \sum_{i=1}^{m} a_{ij} y_i + \sum_{i=1}^{m} a_{ij} \rho_i \geqslant \sigma_j \qquad j = 1, \ldots, n \\ & \qquad y_i \geqslant 0 \qquad\qquad i = 1, \ldots, m \end{aligned}$$

Uma vez que $b_i \rho_i$ é uma constante para todo i, o problema pode ser expresso na forma abaixo:

$$\begin{aligned} P_3: \quad & \text{Minimize} \quad \sum_{i=1}^{m} b_i y_i \\ & \text{Sujeito a :} \\ & \qquad \sum_{i=1}^{m} a_{ij} y_i \geqslant c_j \qquad j = 1, \ldots, n \\ & \qquad y_i \geqslant 0 \qquad\qquad i = 1, \ldots, m \end{aligned}$$

Veremos na seqüência que P_3 é o problema dual de P_0.

6.2 O Problema de Programação Linear

Programação linear (PL) envolve a escolha de valores para variáveis de forma ótima. As variáveis $\{x_j\}$ são ditas variáveis de decisão. O objetivo em PL é sempre maximizar (minimizar) uma função linear:

$$\text{Minimize } c_1 x_1 + c_2 x_2 + \ldots + c_n x_n$$

sob restrições da forma:

$$a_1 x_1 + a_2 x_2 + \ldots + a_n x_n \left\{ \begin{array}{c} \leqslant \\ = \\ \geqslant \end{array} \right\} b$$

É fácil converter uma restrição do tipo "$\leqslant$" em outra equivalente do tipo "=":

$$a_1x_1 + a_2x_2 + \ldots + a_nx_n \leqslant b \Leftrightarrow \begin{cases} a_1x_1 + a_2x_2 + \ldots + a_nx_n + s = b \\ s \geqslant 0 \end{cases}$$

$$a_1x_1 + a_2x_2 + \ldots + a_nx_n \geqslant b \Leftrightarrow \begin{cases} a_1x_1 + a_2x_2 + \ldots + a_nx_n - s = b \\ s \geqslant 0 \end{cases}$$

A variável s é dita variável de folga (*slack variable*), já que seu valor corresponde à quantidade de recurso b não utilizada. Podemos também converter uma restrição do tipo "=" em restrições equivalentes do tipo "$\leqslant$":

$$a_1x_1 + a_2x_2 + \ldots + a_nx_n = b \Leftrightarrow \begin{cases} a_1x_1 + a_2x_2 + \ldots + a_nx_n \leqslant b \\ a_1x_1 + a_2x_2 + \ldots + a_nx_n \geqslant b \end{cases}$$

Podemos ainda converter variáveis irrestritas em sinal em variáveis não negativas:

$$x \in \mathbb{R} \Leftrightarrow \begin{cases} x = x^+ - x^- \\ x^+ \geqslant 0 \\ x^- \geqslant 0 \end{cases}$$

Portanto, podemos assumir que qualquer problema em PL pode ser especificado (re-expresso) na forma abaixo:

$$\begin{array}{lll} PL: & \text{Maximize} & c_1x_1 + c_2x_2 + \ldots + c_nx_n \\ & \text{Sujeito a :} & \\ & & \begin{array}{lcl} a_{1,1}x_1 + a_{1,2}x_2 + \ldots + a_{1,n}x_n & \leqslant & b_1 \\ a_{2,1}x_1 + a_{2,2}x_2 + \ldots + a_{2,n}x_n & \leqslant & b_2 \\ \vdots & \leqslant & \vdots \\ a_{m,1}x_1 + a_{m,2}x_2 + \ldots + a_{m,n}x_n & \leqslant & b_m \\ x_1, \ldots, x_n \geqslant 0 & & \end{array} \end{array}$$

Um vetor com valores para as variáveis de decisão $x = (x_1 \ \ldots \ x_n)^T$ é dito uma solução candidata. A solução x é dita factível se satisfaz as restrições. A solução x é dita ótima se ela é factível e induz o valor ótimo da função objetivo.

Exemplo de Problema Infactível

Um problema é dito infactível se não existe uma solução candidata $x \in \mathbb{R}^n$ que satisfaça às restrições. O problema abaixo é um exemplo de problema infactível, $S = \{x \in \mathbb{R}^n : Ax \leqslant b, x \geqslant 0\} = \emptyset$, sendo este ilustrado na Figura 6.1.

$$\begin{array}{ll} \text{Maximize} & 5x_1 + 4x_2 \\ \text{Sujeito a :} & \\ & \begin{array}{rcrcr} x_1 & + & x_2 & \leqslant & 1 \\ -2x_1 & - & 2x_2 & \leqslant & -9 \\ x_1, x_2 \geqslant 0 & & & & \end{array} \end{array}$$

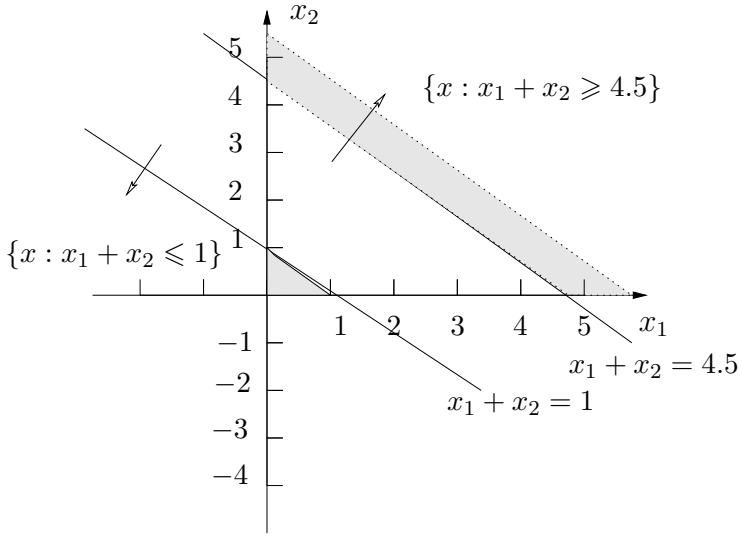


Figura 6.1: Exemplo de problema infactível. As regiões achuradas indicam as regiões factíveis para cada uma das restrições. Observe que a interseção das regiões achuradas é vazia.

Exemplo de Problema Ilimitado

Um problema é ilimitado se não existe um limite superior para o valor da função objetivo, ou seja, podemos crescer o valor da função objetivo arbitrariamente. Abaixo segue um exemplo de problema ilimitado cuja ilustração é feita na Figura 6.2.

$$
\begin{array}{ll}
\text{Maximize} & x_1 - 4x_2 \\
\text{Sujeito a :} & \\
& \begin{array}{rcrcl}
-2x_1 & + & x_2 & \leqslant & -1 \\
-x_1 & - & 2x_2 & \leqslant & -2
\end{array} \\
& x_1, x_2 \geqslant 0
\end{array}
$$

6.3 Algoritmo Simplex

O algoritmo Simplex, proposto por George Dantzig nos anos 40, constituiu um grande avanço científico-tecnológico e deu grande impulso ao campo da pesquisa operacional que estava dando os primeiros passos. O nome do algoritmo tem suas raízes no conceito de simplex: um plano que corta os vetores unitários. O algoritmo como é conhecido atualmente difere da versão original e tem servido de base para versões estendidas para tarefas específicas como, por exemplo, o método dual Simplex que é amplamente adotado em implementações *branch-and-bound* e *branch-and-cut* para resolução de problemas inteiros, e o método Simplex adaptado para o problema de fluxo em redes de custo mínimo.

O algoritmo Simplex pode ser visto como um processo combinatório que busca encontrar as colunas da matriz de restrições que induzem uma base e, portanto, uma solução básica ótima. A dificuldade advém do fato que tipicamente existe um número

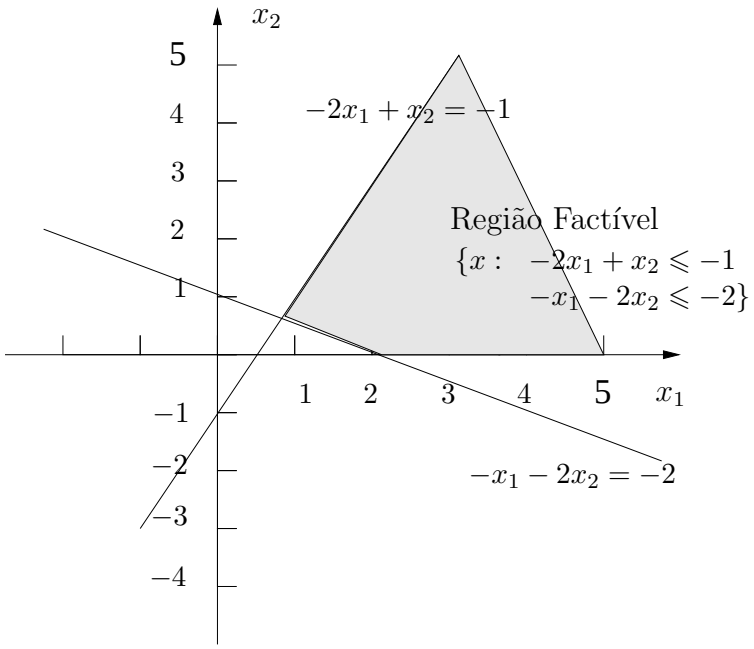


Figura 6.2: Exemplo de problema ilimitado. Podemos avançar dentro da região factível de maneira a crescer o valor da função objetivo sem limites. Por exemplo, ao longo da direção $(x_1, 0)$ podemos avançar sem limites, fazendo o valor da função objetivo aumentar.

exponencial de possíveis combinações de colunas, gerando portando um desempenho de pior caso de ordem exponencial. Apesar deste aspecto desfavorável, o algoritmo Simplex é eficaz e para muitas instâncias continua sendo o algoritmo mais rápido, mesmo quando comparado com algoritmos de ponto-interior que tem desempenho polinomial no pior caso.

No que segue apresentamos o algoritmo Simplex através de um exemplo. Na seqüência daremos detalhes do algoritmo e trataremos aspectos de inicialização.

6.3.1 Exemplo

Considere o problema de programação linear abaixo:

$$\begin{array}{lll} \text{Maximize} & 5x_1 + 4x_2 + 3x_3 & \\ \text{Sujeito a :} & & \\ & 2x_1 + 3x_2 + x_3 \leqslant 5 & \\ & 4x_1 + x_2 + 2x_3 \leqslant 11 & \\ & 3x_1 + 4x_2 + 2x_3 \leqslant 8 & \\ & x_1, x_2, x_3 \geqslant 0 & \end{array} \tag{6.4}$$

Após adicionarmos variáveis de folga, o problema toma a forma a seguir:

$$\begin{array}{lrcrcrcrcr} \text{Maximize} & \delta & = & 0 & + & 5x_1 & + & 4x_2 & + & 3x_3 \\ \text{Sujeito a :} & & & & & & & & & \\ & w_1 & = & 5 & - & 2x_1 & - & 3x_2 & - & x_3 \\ & w_2 & = & 11 & - & 4x_1 & - & x_2 & - & 2x_3 \\ & w_3 & = & 8 & - & 3x_1 & - & 4x_2 & - & 2x_3 \\ & x_1, x_2, x_3, w_1, w_2, w_3 \geqslant 0 & & & & & & & & \end{array} \tag{6.5}$$

O sistema (6.5) coloca o problema de programação linear em uma forma conhecida por "dicionário", na qual a função objetivo e um subconjunto de variáveis (*variáveis básicas*) com cardinalidade igual ao número de restrições são expressos em termos das variáveis restantes (*variáveis não básicas*). As variáveis não básicas assumem valores nulos e, portanto, uma solução pode ser obtida diretamente a partir do dicionário. Para o dicionário acima, a base é formada pelas variáveis w_1, w_2 e w_3, enquanto que as variáveis não básicas são x_1, x_2 e x_3. Já que as variáveis não básicas assumem valores nulos, obtemos diretamente os valores das variáveis básicas: $w_1 = 5$, $w_2 = 11$ e $w_3 = 8$. Note que a solução obtida é factível para o problema em questão uma vez que todas as variáveis (básicas e não básicas) são não negativas. Além disso, a função objetivo corrente tem valor $\delta = 0$.

O método Simplex é um processo iterativo que inicia com uma solução $y^0 = (x_1^0 \ \ldots \ x_n^0 \ w_1^0 \ \ldots \ w_m^0)^T$, onde $n = 3$ e $m = 3$, satisfazendo as equações de (6.4). Partindo do dicionário (6.5), o Simplex busca uma nova solução y^1 tal que: $5x_1^1 + 4x_2^1 + 3x_3^1 > 5x_1^0 + 4x_2^0 + 3x_3^0$. Para tanto, é necessário fazer uma solução não básica com coeficiente positivo na equação de δ entrar na base que, por sua vez, fará a variável não básica aumentar e consequentemente elevar o valor da função objetivo. Obviamente, a variável que entra na base não pode aumentar seu valor indefinidamente, a menos que o problema seja ilimitado. A primeira variável básica a se tornar nula passa a ser uma variável não básica na iteração seguinte. O processo então se repete até que se convirja para uma solução ótima ou até que se detecte que o problema é ilimitado.

Abaixo ilustramos o funcionamento do Algoritmo Simplex na solução da instância (6.4), conforme o esboço dado acima.

Algoritmo Simplex

Inicialização: Para iniciarmos o processo, necessitamos de uma solução factível, tal como:

$$x_1^0 = 0, \ x_2^0 = 0, \ x_3^0 = 0, \ w_1^0 = 5, \ w_2^0 = 11, \ w_3^0 = 8$$

Esta solução y^0 induz o valor $\delta = 0$ para a função objetivo.

Passo 1: A solução corrente não é ótima. Qualquer acréscimo no valor de x_1 aumenta o valor de δ. Mas não podemos aumentar o valor de x_1 ilimitadamente já que este está limitado pelas desigualdades abaixo:

$$\left\{ \begin{array}{lcrcl} w_1 & = & 5 - 2x_1 & \geqslant & 0 \\ w_2 & = & 11 - 4x_1 & \geqslant & 0 \\ w_3 & = & 8 - 3x_1 & \geqslant & 0 \end{array} \right. \Rightarrow \left\{ \begin{array}{lcrcr} x_1 & \leqslant & 5/2 & = & 2.5 \\ x_1 & \leqslant & 11/4 & = & 2.75 \\ x_1 & \leqslant & 8/3 & = & 2.667 \end{array} \right.$$

Assim, o valor de x_1 na próxima iteração é o menor dentre $\{5/2, 11/4, 8/3\}$, o que leva a igualdade:

$$x_1 = \frac{5}{2} - \frac{1}{2}w_1 - \frac{3}{2}x_2 - \frac{1}{2}x_3 \tag{6.6}$$

Substituindo-se (6.6) no sistema (6.5) de maneira a transferir-se a variável w_1 para o lado direito, obtém-se:

$$\begin{array}{rcl} x_1 &=& \frac{5}{2} - \frac{1}{2}w_1 - \frac{3}{2}x_2 - \frac{1}{2}x_3 \\ \\ w_2 &=& 11 - 4\left(\frac{5}{2} - \frac{1}{2}w_1 - \frac{3}{2}x_2 - \frac{1}{2}x_3\right) - x_2 - 2x_3 \\ &=& 1 + 2w_1 + 5x_2 \\ \\ w_3 &=& 8 - 3\left(\frac{5}{2} - \frac{1}{2}w_1 - \frac{3}{2}x_2 - \frac{1}{2}x_3\right) - 4x_2 - 2x_3 \\ &=& \frac{1}{2} + \frac{3}{2}w_1 + \frac{1}{2}x_2 - \frac{1}{2}x_3 \\ \\ \delta &=& 5\left(\frac{5}{2} - \frac{1}{2}w_1 - \frac{3}{2}x_2 - \frac{1}{2}x_3\right) + 4x_2 + 3x_3 \\ &=& \frac{25}{2} - \frac{5}{2}w_1 - \frac{7}{2}x_2 + \frac{1}{2}x_3 \end{array} \tag{6.7}$$

Agora, substituindo as equações (6.7) no "dicionário" (6.5), obtém-se o dicionário abaixo:

$$\begin{array}{rrcrrr} \text{Max} & \delta &=& \frac{25}{2} & -\frac{5}{2}w_1 & -\frac{7}{2}x_2 & +\frac{1}{2}x_3 \\ & x_1 &=& \frac{5}{2} & -\frac{1}{2}w_1 & -\frac{3}{2}x_2 & -\frac{1}{2}x_3 \\ & w_2 &=& 1 & +2w_1 & +5x_2 & \\ & w_3 &=& \frac{1}{2} & +\frac{3}{2}w_1 & +\frac{1}{2}x_2 & -\frac{1}{2}x_3 \end{array} \tag{6.8}$$

A solução induzida pelo dicionário (6.8) é $y^1 = (x_1^1, x_2^1, x_3^1, w_1^1, w_2^1, w_3^1) = (\frac{5}{2}, 0, 0, 0, 1, \frac{1}{2})$ cujo valor da função objetivo é $\delta = \frac{25}{2}$. Neste dicionário, as variáveis x_1, w_2, e w_3 são ditas variáveis básicas tal que o conjunto $\mathbb{B} = \{x_1, w_2, w_3\}$ contém as variáveis básicas. As demais variáveis são ditas não básicas, sendo o conjunto $\mathbb{N} = \{x_2, x_3, w_1\}$ o conjunto das variáveis não básicas.

Passo 2: A solução corrente não é ótima! Note que um pequeno acréscimo no valor de x_3 invariavelmente aumenta o valor de δ. Mas não podemos aumentar o valor de x_3 ilimitadamente uma vez que isto poderia tornar a solução infactível (outras variáveis poderiam assumir valores negativos). Para que a solução resultante seja factível, as desigualdades abaixo devem ser respeitadas:

$$\left\{ \begin{array}{rcrcl} x_1 &=& \frac{5}{2} - \frac{1}{2}x_3 &\geqslant& 0 \\ w_3 &=& \frac{1}{2} - \frac{1}{2}x_3 &\geqslant& 0 \end{array} \right. \Rightarrow \left\{ \begin{array}{l} x_3 \leqslant 5 \\ x_3 \leqslant 1 \end{array} \right.$$

Portanto, w_3 deve sair da base para que a variável x_3 possa entrar na base sem violar as restrições. Após substituirmos a equação $x_3 = 1 + 3w_1 + x_2 - 2w_3$ nas equações do dicionário (6.8), obtemos:

$$\begin{array}{rcl} x_1 &=& \frac{5}{2} - \frac{1}{2}w_1 - \frac{3}{2}x_2 - \frac{1}{2}\left(1 + 3w_1 + x_2 - 2w_3\right) \\ &=& 2 - 2w_1 - 2x_2 + w_3 \\ \\ \delta &=& \frac{25}{2} - \frac{5}{2}w_1 - \frac{7}{2}x_2 + \frac{1}{2}\left(1 + 3w_1 + x_2 - 2w_3\right) \\ &=& 13 - w_1 - 3x_2 - w_3 \end{array} \tag{6.9}$$

Substituindo as equações do dicionário (6.8) pelas equações (6.9) obtemos um novo dicionário:

$$\begin{array}{rrrrrrr} \text{Max} & \delta & = & 13 & -w_1 & -3x_2 & -w_3 \\ & x_1 & = & 2 & -2w_1 & -2x_2 & +w_3 \\ & w_2 & = & 1 & +2w_1 & +5x_2 & \\ & x_3 & = & 1 & +3w_1 & +x_2 & -2w_3 \end{array} \tag{6.10}$$

cuja base é $\mathbb{B} = \{x_1, x_3, w_2\}$. A solução dada pelo dicionário (6.10) é ótima: $x_1 = 2$, $x_2 = 0$, $x_3 = 1$, $w_1 = 0$, $w_2 = 1$, $w_3 = 0$, e $\delta = 13$ é uma solução ótima pois os coeficientes das variáveis não básicas na equação de δ, no dicionário correspondente dado pelo sistema (6.10), são todos negativos; aumentando o valor de qualquer varíavel não básica reduzirá o valor da função objetivo.

6.3.2 Algoritmo Simplex em detalhes

Nesta seção generalizamos os passos ilustrados acima para o caso geral do problema de programação linear. Considere a forma geral do problema de programação linear:

$$\begin{array}{ll} \text{Maximize} & \sum_{j=1}^{n} c_j x_j \\ \text{Sujeito a :} & \\ & \sum_{j=1}^{n} a_{ij} x_j \leqslant b_i \qquad i = 1, \ldots, m \\ & x_j \geqslant 0 \qquad\qquad\quad j = 1, \ldots, n \end{array} \tag{6.11}$$

Inicialização: Obter Dicionário Inicial

Inicialmente é necessário introduzir variáveis de folga na formulação (6.11) como segue:

$$\begin{array}{rcll} \delta & = & \sum_{j=1}^{n} c_j x_j & \\ w_i & = & b_i - \sum_{j=1}^{n} a_{ij} x_j \qquad & i = 1, \ldots, m \end{array} \tag{6.12}$$

Com o intuito de facilitar os próximos desenvolvimentos, vamos assumir que $w_i = x_{n+i}$ para $i = 1, \ldots, m$, dessa forma gerando o dicionário:

$$\begin{array}{rcll} \delta & = & \sum_{j=1}^{n} c_j x_j & \\ x_{n+i} & = & b_i - \sum_{j=1}^{n} a_{ij} x_j \qquad & i = 1, \ldots, m \end{array} \tag{6.13}$$

cuja base é dada pelas colunas correspondentes às variáveis do conjunto $\{x_{n+1}, \ldots, x_{n+m}\}$. Seja $\mathcal{B} \subseteq \{1, \ldots, n+m\}$ o conjunto com os índices das variáveis básicas. Seja $\mathcal{N} = \{1, \ldots, n+m\} - \mathcal{B}$ o conjunto com os índices das variáveis não básicas. Inicialmente, vamos assumir que $x_1 = 0, \ldots, x_n = 0$ induz uma solução

factível, o que equivale a dizer que $b_i \geqslant 0$ para $i = 1, \ldots, m$. Durante a aplicação do algoritmo Simplex, o dicionário assumirá a forma:

$$\begin{aligned} \delta &= \bar{\delta} + \sum_{j \in \mathcal{N}} \bar{c}_j x_j \\ x_i &= \bar{b}_i - \sum_{j \in \mathcal{N}} \bar{a}_{ij} x_j, \qquad \forall i \in \mathcal{B} \end{aligned} \tag{6.14}$$

onde $\bar{c}_j$ é conhecido por custo reduzido da variável não básica x_j.

Laço de Iteração

No que segue descrevemos os passos do algoritmo Simplex.

i. Escolha um $k \in \{j \in \mathcal{N} : \bar{c}_j > 0\}$. Se não existir nenhum k com tal propriedade, então a solução corrente é ótima. A variável x_k é dita *variável que entra na base*.

ii. A variável que sairá da base é escolhida de forma que a solução subseqüente continue viável.

$$\begin{aligned} x_i = \bar{b}_i - \bar{a}_{ik} x_k \quad \forall i \in \mathcal{B} \quad &\Rightarrow \quad \bar{b}_i - \bar{a}_{ik} x_k \geqslant 0 \quad \forall i \in \mathcal{B} \\ &\Rightarrow \quad \bar{a}_{ik} x_k \leqslant \bar{b}_i \qquad \forall i \in \mathcal{B} \end{aligned}$$

Considerando a última desigualdade, surgem três casos:

(a) Se não existe $\bar{a}_{ik} > 0,\ i \in \mathcal{B}$, então o problema é ilimitado (*unbounded*).

(b) Se existe $\bar{b}_i = 0$ e $\bar{a}_{ik} > 0$ para algum $i \in \mathcal{B}$, então x_i é a variável que sai da base e o pivoteamento é dito *degenerado*. Pivoteamento compreende todos os passos envolvidos em uma mudança de base. O pivoteamento é dito degenerado porque uma variável entrou na base com valor nulo e, conseqüentemente, não aumentou o valor da função objetivo.

(c) Se os dois casos acima não ocorrem, então a variável que sai da base é $l \in \{i \in \mathcal{B} : \bar{a}_{ik}/\bar{b}_i$ é máximo, $\bar{a}_{ik} > 0$ e $\bar{b}_i > 0\}$. Isto é, $l \in \{i \in \mathcal{B} : \bar{b}_i/\bar{a}_{ik}$ é mínimo$\}$.

iii. Uma vez escolhida a variável x_k que entra na base e a variável x_l que sai da base, executamos as operações apropriadas nas linhas do dicionário. A iteração é chamada de *pivoteamento*. Repetimos a partir do primeiro passo até que uma solução ótima seja atingida.

Não ocorrendo pivoteamentos degenerados o algoritmo Simplex converge em tempo finito para a solução ótima, ou este detecta que o problema é ilimitado. Na presença de pivoteamentos degenerados o algoritmo Simplex pode entrar em ciclos e nunca convergir. Para que isto seja evitado, regras de pivoteamento foram concebidas que garantem convergência em um número exponencial no número de colunas da matriz. Para maiores detalhes sobre regras de pivoteamento, o leitor pode consultar os textos de Chvatal [11] e Vanderbei [42]. Uma regra simples que garante convergência é a regra de Bland (*Bland's rule*). Segundo esta regra, a variável que entra na base e a variável que sai da base, quando existir mais do que uma alternativa, são aquelas que tiverem menor índice dentre os seus respectivos conjuntos de alternativas.

6.3.3 Inicialização

Até este momento, consideramos problemas cujos b_i's são todos não negativos. Isso permitia a obtenção de uma solução viável fazendo $x_j = 0$ para $j = 1, \ldots, n$ e $x_{i+n} = b_i$ para $i = 1, \ldots, m$. *O que devemos fazer se algum b_i é negativo*? Contornamos esta dificuldade ao resolvermos um problema auxiliar, cuja solução inicial é viável e cuja solução ótima é viável para o problema original, isto se este é factível. O problema auxiliar é expresso como segue:

$$\begin{array}{lll} P_A: & \text{Maximize} & -x_0 \\ & \text{Sujeito a:} & \\ & & \sum_{j=1}^{n} a_{ij}x_j - x_0 \leqslant b_i \quad i = 1, \ldots, m \\ & & x_j \geqslant 0 \qquad\qquad\quad j = 0, \ldots, n \end{array} \tag{6.15}$$

Não é difícil de ver que o problema (6.11) possui uma solução factível se e somente se a solução ótima do problema auxiliar (6.15) tem valor nulo como valor ótimo da função objetivo. É fácil de se obter uma solução para (6.15): basta definir $x_j = 0$ para $j = 1, \ldots, n$ e tomar um valor suficientemente grande para x_0.

Exemplo

Aqui ilustramos o emprego do problema auxiliar na busca de uma solução factível para um problema de programação linear.

Problema Original: Considere como problema de programação linear a formulação:

$$\begin{array}{lll} \text{Maximize} & -2x_1 - x_2 & \\ \text{Sujeito a:} & & \\ & -x_1 + x_2 \leqslant -1 & \\ & -x_1 - 2x_2 \leqslant -2 & \\ & x_2 \leqslant 1 & \\ & x_1, x_2 \geqslant 0 & \end{array}$$

Problema Auxiliar: De acordo com os desenvolvimentos anteriores, o problema auxiliar toma a forma:

$$\begin{array}{lll} \text{Maximize} & -x_0 & \\ \text{Sujeito a:} & & \\ & -x_1 + x_2 - x_0 \leqslant -1 & \\ & -x_1 - 2x_2 - x_0 \leqslant -2 & \\ & x_2 - x_0 \leqslant 1 & \\ & x_0, x_1, x_2 \geqslant 0 & \end{array}$$

Dicionário Inicial: Tomando o dicionário inicial como se os elementos b_i's fossem todos não negativos, obtemos:

$$\begin{array}{lrcrrr} \text{Max} & \delta & = & & & -x_0 \\ & w_1 & = & -1 & +x_1 - x_2 & +x_0 \\ & w_2 & = & -2 & +x_1 + 2x_2 & +x_0 \\ & w_3 & = & 1 & - x_2 & +x_0 \end{array}$$

Para tornar o dicionário viável, basta executar um pivoteamento com a variável x_0 fazendo a variável básica mais negativa sair na base. Assim x_0 é a variável que entra na base enquanto que w_2 é a variável que sai da base. As equações a seguir descrevem a operação de pivoteamento.

$$\begin{aligned} x_0 &= 2 - x_1 - 2x_2 + w_2 \\ \\ w_1 &= -1 + x_1 - x_2 + (2 - x_1 - 2x_2 + w_2) \\ &= 1 - 3x_2 + w_2 \\ \\ w_3 &= 1 - x_2 + x_0 \\ &= 1 - x_2 + (2 - x_1 - 2x_2 + w_2) \\ &= 3 - x_1 - 3x_2 + w_2 \end{aligned} \tag{6.16}$$

Executando o pivoteamento conforme equações (6.16), obtemos o dicionário abaixo:

$$\begin{array}{llllll} \text{Max} & \delta &= -2 & +x_1 & +2x_2 & -w_2 \\ & w_1 &= 1 & & -3x_2 & +w_2 \\ & x_0 &= 2 & -x_1 & -2x_2 & +w_2 \\ & w_3 &= 3 & -x_1 & -3x_2 & +w_2 \end{array}$$

Segundo Pivoteamento: Fazendo x_1 entrar na base, x_0 deve obrigatoriamente deixar a base, gerando o dicionário:

$$\begin{array}{llllll} \text{Max} & \delta &= 0 & -x_0 & & \\ & w_1 &= 1 & & -3x_2 & +w_2 \\ & x_1 &= 2 & -x_0 & -2x_2 & +w_2 \\ & w_3 &= 1 & +x_0 & -x_2 & \end{array}$$

Note que o dicionário acima é ótimo para o problema auxiliar e, mais ainda, a solução candidata $x_1 = 2$, $x_2 = 0$, $w_1 = 1$, $w_2 = 0$ e $w_3 = 1$ é factível para o problema original dado que $\delta = 0$.

Obtendo Dicionário Factível: Eliminando a coluna de x_0 e introduzindo a função objetivo inicial, obtemos um dicionário factível para o problema original:

$$\begin{array}{llllll} \text{Max} & \delta &= -4 & +3x_2 & -2w_2 \\ & w_1 &= 1 & -3x_2 & +w_2 \\ & x_1 &= 2 & -2x_2 & +w_2 \\ & w_3 &= 1 & -x_2 & \end{array}$$

Pivoteamento: O dicionário acima não é ótimo, assim podemos fazer x_2 entrar na base e w_1 sair da base, obtendo o dicionário:

$$\begin{array}{llllll} \text{Max} & \delta &= -3 & -w_1 & -w_2 \\ & x_2 &= \frac{1}{3} & -\frac{w_1}{3} & +\frac{w_2}{3} \\ & x_1 &= \frac{4}{3} & +\frac{2}{3}w_1 & +\frac{1}{3}w_2 \\ & w_3 &= \frac{2}{3} & +\frac{w_1}{3} & -\frac{w_2}{3} \end{array}$$

Portanto, encontramos uma solução ótima do problema original, onde $x_1 = \frac{4}{3}$ e $x_2 = \frac{1}{3}$, com valor objetivo igual a -3.

6.4 Dualidade

Associado a um problema de programação linear P (primal) está outro problema de programação linear chamado de dual e denotado por D. Há várias conseqüências teóricas e práticas resultantes da teoria da dualidade. Por exemplo, qualquer solução para D induz um limite para o valor ótimo da função objetivo de P, o problema primal, e vice-versa.

6.4.1 Motivação

Considere o problema exemplo abaixo:

$$\begin{array}{lll} P: \text{ Maximize} & 4x_1 + x_2 + 3x_3 & \\ \text{Sujeito a :} & & \\ & x_1 + 4x_2 \leqslant 1 & \\ & 3x_1 - x_2 + x_3 \leqslant 3 & \\ & x_1, x_2, x_3 \geqslant 0 & \end{array} \tag{6.17}$$

Qualquer solução factível para P induz um limite inferior. Por exemplo, $x' = (1, 0, 0)$ nos diz que o valor ótimo da função objetivo $\delta^* \geqslant 4$. Usando a solução $x'' = (0, 0, 3)$ descobrimos que $\delta^* \geqslant 9$. Quão próximo do ótimo estão estes limites inferiores? Vamos multiplicar a primeira restrição de (6.17) por 2, multiplicar a segunda restrição por 3, e depois adicioná-las como segue:

$$\begin{array}{rcl} 2(x_1 + 4x_2) & \leqslant & 2(1) \\ 3(3x_1 - x_2 + x_3) & \leqslant & 3(3) \\ \hline 11x_1 + 5x_2 + 3x_3 & \leqslant & 11 \end{array}$$

Note que $4x_1 + x_2 + 3x_3 \leqslant 11x_1 + 5x_2 + 3x_3 \leqslant 11$, pois $x_j \geqslant 0$. Assim concluímos que $9 \leqslant \delta^* \leqslant 11$. Para obtermos limites superiores mais apertados, utilizamos o mesmo procedimento mas desta vez com variáveis em vez de números específicos. Multiplicamos as restrições por duas variáveis não negativas:

$$\begin{array}{rcl} y_1(x_1 + 4x_2) & \leqslant & y_1 \\ y_2(3x_1 - x_2 + x_3) & \leqslant & 3y_2 \\ \hline (y_1 + 3y_2)x_1 + (4y_1 - y_2)x_2 + y_2x_3 & \leqslant & y_1 + 3y_2 \end{array}$$

Agora, estipulamos que:

$$\begin{array}{rcl} y_1 + 3y_2 & \geqslant & 4 \\ 4y_1 - y_2 & \geqslant & 1 \\ y_2 & \geqslant & 3 \\ y_1, y_2 & \geqslant & 0 \end{array} \tag{6.18}$$

Valores para (y_1, y_2) satisfazendo as desigualdades (6.18) nos levam às seguintes desigualdades:

$$\begin{array}{rcl} \delta & = & 4x_1 + x_2 + 3x_3 \\ & \leqslant & (y_1 + 3y_2)x_1 + (4y_1 - y_2)x_2 + y_2x_3 \\ & \leqslant & y_1 + 3y_2 \end{array}$$

Dessa maneira obtemos o limite superior $(y_1 + 3y_2)$ para δ^*. Nossa tarefa é minimizar este limite superior através da solução do problema de programação linear que segue:

$$\begin{array}{lll} D: \text{ Minimize} & y_1 + 3y_2 & \\ \text{Sujeito a :} & & \\ & y_1 + 3y_2 & \geqslant 4 \\ & 4y_1 - y_2 & \geqslant 1 \\ & y_2 & \geqslant 3 \\ & y_1, y_2 & \geqslant 0 \end{array}$$

6.4.2 O Problema Dual

Aqui desenvolvemos a maneira através da qual o problema dual pode ser obtido a partir do problema primal. Primeiramente, considere o problema primal que segue:

$$\begin{array}{lll} P: \text{ Maximize} & \sum_{j=1}^{n} c_j x_j & \\ \text{Sujeito a :} & & \\ & \sum_{j=1}^{n} a_{ij} x_j \leqslant b_i & i = 1, \ldots, m \\ & x_j \geqslant 0 & j = 1, \ldots, n \end{array}$$

O problema dual associado a P é:

$$\begin{array}{lll} D: \text{ Minimize} & \sum_{i=1}^{m} b_i y_i & \\ \text{Sujeito a :} & & \\ & \sum_{i=1}^{m} a_{ij} y_i \geqslant c_j & j = 1, \ldots, n \\ & y_i \geqslant 0 & i = 1, \ldots, m \end{array}$$

Queremos mostrar que o dual do dual é o problema primal. Podemos expressar o dual como segue:

$$\begin{array}{lll} D: \text{ Maximize} & -\sum_{i=1}^{m} b_i y_i & \\ \text{Sujeito a :} & & \\ & \sum_{i=1}^{m} -a_{ij} y_i \leqslant -c_j & j = 1, \ldots, n \\ & y_i \geqslant 0 & i = 1, \ldots, m \end{array}$$

Gerando o dual de D, obtemos:

$$\begin{array}{lll} DD: \text{ Minimize} & \sum_{j=1}^{n} -c_j x_j & \\ \text{Sujeito a :} & & \\ & \sum_{j=1}^{n} -a_{ij} x_j \geqslant -b_i & i = 1, \ldots, m \\ & x_j \geqslant 0 & j = 1, \ldots, n \end{array}$$

que por sua vez pode ser colocado na forma:

$$\begin{array}{lll} DD: \text{ Maximize } & \sum_{j=1}^{n} c_j x_j & \\ \text{Sujeito a :} & & \\ & \sum_{j=1}^{n} a_{ij} x_j \leqslant b_i & i = 1, \ldots, m \\ & x_j \geqslant 0 & j = 1, \ldots, n \end{array}$$

Os desenvolvimentos acima demonstram que $DD \equiv P$.

6.4.3 O Teorema Fraco da Dualidade

Teorema 6.1 *(Dualidade Fraca) Seja $x = (x_1 \ \ldots \ x_n)^T$ uma solução primal factível e $y = (y_1 \ \ldots y_m)^T$ uma solução dual factível. Então:*

$$c^T x = \sum_{j=1}^{n} c_j x_j \leqslant \sum_{i=1}^{m} b_i y_i = b^T y$$

Prova:

$$\begin{array}{rcll} \sum_{j=1}^{n} c_j x_j & \leqslant & \sum_{j=1}^{n} \left(\sum_{i=1}^{m} a_{ij} y_i \right) x_j & \text{pois } x_j \geqslant 0 \text{ e } \sum_{i=1}^{m} a_{ij} y_i \geqslant c_j \\ & = & \sum_{i=1}^{m} \left(\sum_{j=1}^{n} a_{ij} x_j \right) y_i & \\ & \leqslant & \sum_{i=1}^{m} b_i y_i & \text{pois } \sum_{j=1}^{n} a_{ij} x_j \leqslant b_i \text{ e } y_i \geqslant 0 \end{array}$$

■

Se obtemos uma solução primal $x^* = (x_1^* \ \ldots \ x_n^*)^T$ e uma dual $y^* = (y_1^* \ \ldots \ y_m^*)^T$ ambas factíveis, tal que

$$\sum_{j=1}^{n} c_j x_j^* = \sum_{i=1}^{m} b_i y_i^*$$

então podemos concluir que ambas as soluções são ótimas para seus respectivos problemas. Para o problema exemplo dado por (6.17) as soluções ótimas são $x^* = (0 \ 0.25 \ 3.25)^T$ e $y^* = (1 \ 3)^T$.

6.4.4 O Teorema Forte da Dualidade

O fato de que em programação linear não há folga entre o valor ótimo da função objetivo dos problemas primal e dual é conhecido como Teorema Forte da Dualidade.

Teorema 6.2 *(Dualidade Forte) Se o problema primal admite uma solução ótima x^* e o dual uma solução ótima y^*, então:*

$$c^T x^* = \sum_{j=1}^{n} c_j x_j^* = \sum_{i=1}^{m} b_i y_i^* = b^T y^*.$$

Ilustração do Teorema Forte

Abaixo seguem os problemas primal e dual.

$$
\begin{array}{lllll}
P: & \text{Max} & 4x_1 + x_2 + 3x_3 & & \\
& S.a: & & & \\
& & x_1 + 4x_2 & \leqslant & 1 \\
& & 3x_1 - x_2 + x_3 & \leqslant & 3 \\
& & x_1, x_2, x_3 & \geqslant & 0
\end{array}
\qquad
\begin{array}{lllll}
D: & \text{Min} & y_1 + 3y_2 & & \\
& S.a: & & & \\
& & y_1 + 3y_2 & \geqslant & 4 \\
& & 4y_1 - y_2 & \geqslant & 1 \\
& & y_2 & \geqslant & 3 \\
& & y_1, y_2 & \geqslant & 0
\end{array}
$$

Os dicionários iniciais são:

$$
\begin{array}{lllllll}
\text{Max} & \delta & = & 0 & +4x_1 & +x_2 & +3x_3 \\
& w_1 & = & 1 & -x_1 & -4x_2 & \\
& w_2 & = & 3 & -3x_1 & +x_2 & -x_3
\end{array}
\qquad
\begin{array}{lllll}
\text{Max} & -\delta & = & 0 & -y_1 & -3y_2 \\
& z_1 & = & -4 & +y_1 & +3y_2 \\
& z_2 & = & -1 & +4y_1 & -y_2 \\
& z_3 & = & -3 & & +y_2
\end{array}
$$

Os dicionários acima podem ser representados por duas matrizes, as matrizes P e D abaixo:

$$
P = \begin{bmatrix} 0 & 4 & 1 & 3 \\ 1 & -1 & -4 & 0 \\ 3 & -3 & 1 & -1 \end{bmatrix}
\qquad
D = \begin{bmatrix} 0 & -1 & -3 \\ -4 & 1 & 3 \\ -1 & 4 & -1 \\ -3 & 0 & 1 \end{bmatrix}
$$

Note que $D = -P^T$. Se aplicarmos o método Simplex ao problema primal e, ao mesmo tempo, executarmos pivoteamento análogo no dual podemos verificar que a propriedade $D = -P^T$ será mantida. Mais especificamente, considere uma solução primal factível a qual faremos x_3 entrar na base e w_2 a deixar a base. Uma vez que as colunas do primal correspondem às linhas do dual, concluímos que a "coluna x_3" do primal corresponde à "linha z_3" do dual. Da mesma forma, a "linha w_2" do primal corresponde à "coluna y_2" do dual. Portanto, a variável y_2 entra na base do dual enquanto que z_3 deixa a base.

Fazendo x_3 entrar na base e w_2 deixar a base, obtemos os dicionários abaixo:

$$
\begin{array}{lllllll}
\text{Max} & \delta & = & 9 & -5x_1 & +4x_2 & -3w_2 \\
& w_1 & = & 1 & -x_1 & -4x_2 & \\
& x_3 & = & 3 & -3x_1 & +x_2 & -w_2
\end{array}
\qquad
\begin{array}{lllll}
\text{Max} & -\delta & = & -9 & -y_1 & -3z_3 \\
& z_1 & = & 5 & +y_1 & +3z_3 \\
& z_2 & = & -4 & +4y_1 & -z_3 \\
& y_2 & = & 3 & & +z_3
\end{array}
$$

Deixando x_2 entrar na base e removendo w_1, produzimos os dicionários abaixo:

$$
\begin{array}{lllllll}
(Primal): & \text{Max} & \delta & = & 10 & -6x_1 & -w_1 & -3w_2 \\
& & x_2 & = & \frac{1}{4} & -\frac{1}{4}x_1 & -\frac{1}{4}w_1 & \\
& & x_3 & = & 3.25 & -3.25x_1 & -\frac{1}{4}w_1 & -w_2
\end{array}
$$

$$
\begin{array}{lllllll}
(Dual): & \text{Max} & -\delta & = & -10 & -0.25z_2 & -3.25z_3 \\
& & z_1 & = & 6 & +\frac{1}{4}z_2 & +3.25z_3 \\
& & y_1 & = & +1 & +\frac{1}{4}z_2 & +\frac{1}{4}z_3 \\
& & y_2 & = & 3 & & +z_3
\end{array}
$$

Ao analisarmos o dicionário do primal, verificamos que a solução é ótima, dado que os coeficientes da linha da função objetivo são todos negativos. Analisando o dicionário do dual verificamos que este é factível e também ótimo. Para obter o dicionário do primal a partir do dicionário do dual, obtenha a matriz transposta negativa e troque x_j por z_j e w_i por y_i. Observe que enquanto o primal não for ótimo, a solução induzida pelo dicionário do dual é infactível.

6.4.5 Folga Complementar

A folga complementar estabelece uma relação algébrica entre a solução primal ótima e a solução dual ótima. Tais condições podem ser obtidas a partir das condições de otimamlidade de primeira ordem para problemas não-lineares (*the Karush-Kuhn-Tucker conditions*), a serem discutidas mais à frente. Para dar uma ideia da importância de tais condições, os algoritmos ponto-interior primal-dual [45] em essência buscam valores para $(x, y, w, z) \in \mathbb{R}^{2(n+m)}$ tal que (x, w) seja primal factível, (y, z) seja dual factível e (x, y, w, z) satisfaça as condições de folga complementar.

Teorema 6.3 *Suponha que* $x = (x_1 \ \dots \ x_n)^T$ *é uma solução primal factível e que* $y = (y_1 \ \dots \ y_n)^T$ *é dual factível. Seja* $w = (w_1 \ \dots \ w_m)^T$ *o vetor com as variáveis de folga do primal e* $z = (z_1 \ \dots \ z_n)^T$ *o vetor com as variáveis de folga do dual. Então* x *e* y *são soluções ótimas para seus respectivos problemas se, e somente se,*

$$\begin{cases} x_j z_j = 0 & \text{para } j = 1, \dots, n \\ y_i w_i = 0 & \text{para } i = 1, \dots, m. \end{cases} \tag{6.19}$$

Fazendo $X = diag(x_1, \dots, x_n)$, $Z = diag(z_1, \dots, z_n)$, $Y = diag(y_1, \dots, y_m)$ e $W = diag(w_1, \dots, w_m)$, as condições (6.19) podem ser expressas como:

$$\begin{aligned} XZe &= 0 \\ YWe &= 0 \end{aligned}$$

onde $e = (1 \ \dots 1)^T$ é um vetor de dimensão apropriada com todas as entradas iguais a 1. Portanto, (x, y, w, z) define soluções primais e duais ótimas se e somente se:

$$\begin{cases} Ax + w & = & b \\ A^T y - z & = & c \\ XZe & = & 0 \\ YWe & = & 0 \\ (x, y, w, z) & \geqslant & 0 \end{cases} \tag{6.20}$$

Para o caso mais simples (mas igualmente geral), onde o problema primal só tem restrições de igualdade ($Ax = b$) enquanto o vetor de decisões é não nulo ($x \geqslant 0$), as condições (6.20) assumem uma forma mais simples:

$$\begin{cases} Ax & = & b \\ A^T y - z & = & c \\ XZe & = & 0 \\ (x, z) & \geqslant & 0 \end{cases} \tag{6.21}$$

Note que se não fossem as restrições $x \geqslant 0$ e as folgas duais $z \geqslant 0$, poderíamos aplicar o método de Newton. O caminho central $\mathcal{C}$ é um arco parametrizado por $\tau > 0$, onde um ponto $(x_\tau, y_\tau, z_\tau) \in \mathcal{C}$ se resolver o sistema:

$$\begin{cases} Ax & = & b \\ A^T y - z & = & c \\ XZe & = & \tau \\ (x, z) & > & 0 \end{cases} \tag{6.22}$$

O caminho central $\mathcal{C} = \{(x_\tau, y_\tau, z_\tau) : \tau > 0\}$ desempenha um papel importante na teoria de algoritmos de ponto interior. As equações (6.22) aproximam as equações (6.21) à medida que τ tende para zero. O caminho central guia a série de soluções candidatas ao longo de uma direção que se afasta de soluções espúrias ao exigir que os produtos $x_i z_i$ sejam estritamente positivos e decrescentes para zero a uma mesma taxa. Muitos métodos primal-dual tomam o passo de Newton na direção de $\mathcal{C}$ para cada $\tau > 0$, em vez de passos de Newton puros que ignoram as desigualdades. Uma vez que estes passos têm tendência para o centro do poliedro definido por $(x, s) \geqslant 0$, tipicamente é possível tomar passos mais longos do que passos de Newton puros sem violar as desigualdades.

6.5 Algoritmo Simplex em Notação Matricial

Após inserirmos variáveis de folga sempre que necessário, podemos assumir a seguinte formulação do problema de programação linear:

$$\begin{array}{lll} \text{Maximize} & c^T x & \\ \text{Sujeito a :} & & \\ & Ax & = \quad b \\ & x & \geqslant \quad 0 \end{array} \tag{6.23}$$

O algoritmo simplex é um processo iterativo, consistindo em escolher variáveis básicas a cada iteração. A Figura 6.3 ilustra as operações de pivoteamento que produzem as iterações $x^0 \to x^1 \to \ldots \to x^*$, tendo como ponto de convergência a solução ótima x^*.

Conforme notação estabelecida acima, $\mathcal{B}$ é o conjunto de índices das variáveis básicas, enquanto $\mathcal{N}$ denota o conjunto dos índices das variáveis não básicas. Assim, cada linha i de $Ax = b$ pode ser escrita como

$$\sum_{j=1}^{n} a_{ij} x_j = b_i \Leftrightarrow \sum_{j \in \mathcal{B}} a_{ij} x_j + \sum_{j \in \mathcal{N}} a_{ij} x_j = b_i.$$

Seja B a submatriz $m \times m$ de $A \in \mathbb{R}^{m \times n}$ correspondente às colunas básicas. Note que B tem posto completo, $rank(B) = m$, daí surge o nome de matriz básica já que esta pode gerar o espaço de todos os possíveis $b \in \mathbb{R}^m$. A dificuldade combinatória de programação linear está em encontrar m colunas dentre $n > m$ colunas que induzem a solução básica ótima: o número de possíveis combinações de colunas é $n!/m!(n-m)!$. Seja N a submatriz $m \times (n-m)$ de A correspondente às colunas não básicas. Então, podemos escrever $Ax = b$ como $Bx_B + Nx_N = b$, onde x_B é o vetor com as variáveis

básicas e x_N, o vetor com as variáveis não básicas. O problema (6.23) pode então ser expresso como:

$$\begin{array}{ll} \text{Maximize} & c_B^T x_B + c_N^T x_N \\ \text{Sujeito a :} & \\ & \begin{array}{rcl} Bx_B + Nx_N & = & b \\ x = (x_B, x_N) & \geqslant & 0 \end{array} \end{array} \tag{6.24}$$

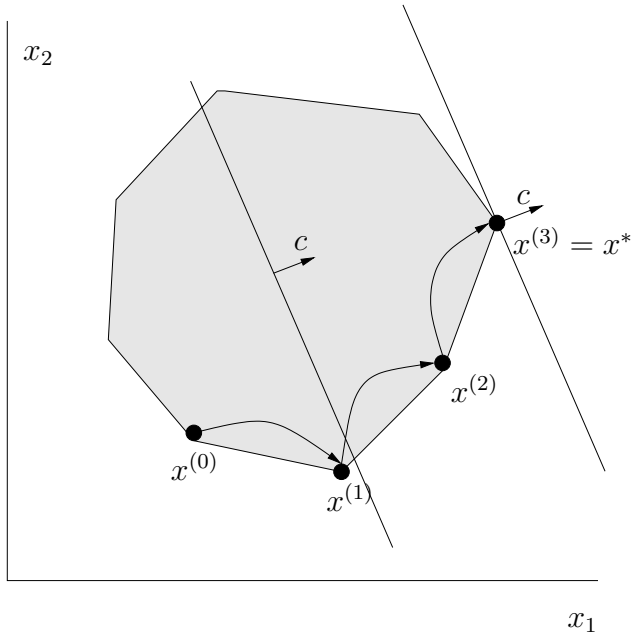


Figura 6.3: Método simplex como um processo iterativo.

6.5.1 Dicionário em Forma Matricial

O dicionário tem a propriedade de que as variáveis básicas podem ser escritas como uma função das variáveis não básicas. Portanto,

$$Bx_B + Nx_N = b \Rightarrow x_B = B^{-1}b - B^{-1}Nx_N. \tag{6.25}$$

Substituindo (6.25) na equação da função objetivo, deduzimos que

$$\begin{array}{rcl} \delta & = & c_B^T x_B + c_N^T x_N = \\ & = & c_B^T \left(B^{-1}b - B^{-1}Nx_N\right) + c_N^T x_N \\ & = & c_B^T B^{-1}b + \left(c_N^T - c_B^T B^{-1}N\right) x_N. \end{array} \tag{6.26}$$

Se combinarmos (6.25) e (6.26), obtemos o dicionário:

$$\begin{array}{rcl} \delta & = & c_B^T B^{-1}b + \left(c_N^T - c_B^T B^{-1}N\right) x_N \\ x_B & = & B^{-1}b - B^{-1}Nx_N. \end{array}$$

6.6 Referências

O material apresentado neste capítulo foi compilado a partir dos textos de Chvatal [11] e Vanderbei [42], que são excelentes referências sobre programação linear. A segunda referência é mais moderna, apresentando aplicações diversas, fazendo uma introdução à programação convexa e também discutindo algoritmos de ponto-interior. Para um estudo mais aprofundado de métodos de ponto-interior recomendamos o livro de Wright [45].

6.7 Exercícios

EX 6.1: Utilize o método Simplex (executando as operações conforme ilustrado em sala de aula) para resolver o problema abaixo:

$$\begin{array}{lll} \text{Maximize} & 2x_1 - 6x_2 & \\ \text{Sujeito a :} & & \\ & -x_1 - x_2 - x_3 & \leqslant \ -2 \\ & 2x_1 - x_2 + x_3 & \leqslant \ 1 \\ & x_1, x_2, x_3 & \geqslant \ 0 \end{array}$$

EX 6.2: Utilize o método Simplex (no papel ou fazendo uso de pacote de otimização linear) para resolver o problema abaixo:

$$\begin{array}{lll} \text{Maximize} & 3x_1 + 2x_2 & \\ \text{Sujeito a :} & & \\ & x_1 - 2x_2 & \leqslant \ 1 \\ & x_1 - x_2 & \leqslant \ 2 \\ & 2x_1 - x_2 & \leqslant \ 6 \\ & x_1 & \leqslant \ 5 \\ & 2x_1 + x_2 & \leqslant \ 16 \\ & x_1 + x_2 & \leqslant \ 12 \\ & x_1 + 2x_2 & \leqslant \ 21 \\ & x_2 & \leqslant \ 10 \\ & x_1, x_2 & \geqslant \ 0 \end{array}$$

EX 6.3: Mostre em um gráfico a região factível do problema dado em EX 6.2. Indique a seqüência de soluções básicas produzida pelo método Simplex.

EX 6.4: Dê um exemplo mostrando que uma variável que se torna básica em uma iteração do Simplex pode se tornar não-básica na iteração seguinte.

EX 6.5: Mostre que uma variável que se torna não-básica em uma iteração do Simplex não pode tornar-se básica na iteração seguinte.

EX 6.6: Encontre o dual do seguinte problema de programação linear:

$$\begin{array}{rll} P: & \text{Maximize} & c^T x \\ & \text{Sujeito a:} & \\ & & a \leqslant Ax \leqslant b \\ & & l \leqslant x \leqslant u \\ & & x \geqslant 0 \end{array}$$

EX 6.7: Ilustre o Teorema Forte da Dualidade no problema dado no exercício EX 6.1. (Se necessário, utilize o pacote de programação linear "lp_solve" disponível no diretório /users/professores/camponog/lp_solve_1.4)

EX 6.8: Para o problema de programação linear dado por $\max\{c^T x : Ax \leqslant b, x \geqslant 0\}$, podemos dizer que as condições de folga complementar (Teorema 6.3) são equivalentes às condições abaixo?

$$\begin{aligned} x^T z &= 0 \\ y^T w &= 0 \end{aligned}$$

EX 6.9: Considere o problema de programação linear dado por:

$$\begin{array}{rll} P_L: & \text{Maximize} & c^T x \\ & \text{Sujeito a:} & Ax = b \\ & & x \in \mathbb{R}^n \end{array}$$

Será que é necessário fazer uso de um algoritmo de programação linear (e.g., Simplex) para resolver P_L?

EX 6.10: O problema de mínimos quadrados consiste em encontrar $x \in \mathbb{R}^n$ tal que minimize $f_2(x) = \|Ax - b\|_2$. Seja P um problema semelhante ao de mínimos quadrados onde utilizamos a norma l-1 e restrições nos valores dos parâmetros:

$$\begin{array}{rll} P: & \underset{x \in \mathbb{R}^n}{\text{Minimize}} & f_1(x) = \|Ax - b\|_1 \\ & \text{Sujeito a:} & x \geqslant l \\ & & x \leqslant u \end{array}$$

onde $A \in \mathbb{R}^{m \times n}$ com $m \gg n$ e $l, u \in \mathbb{R}^n$.

Como que você resolveria P? Escolha um conjunto de $m = 30$ pontos no plano a partir de perturbações de pontos de uma reta conhecida. Ilustre a reta obtida resolvendo um problema de mínimos quadrados e a reta obtida resolvendo P (desconsidere as restrições). (O problema de mínimos quadrados, $\min f_2(x)$, tem solução dada por $x = (A^T A)^{-1} A^T b$.)

EX 6.11: Como você resolveria o problema abaixo P dado por:

$$\begin{array}{rll} P: & \underset{x \in \mathbb{R}^n}{\text{Minimize}} & f_\infty(x) = \|Ax - b\|_\infty \\ & \text{Sujeito a:} & x \geqslant l \\ & & x \leqslant u \end{array}$$

onde $A \in \mathbb{R}^{m \times n}$ com $m \gg n$ e $l, u \in \mathbb{R}^n$.

Capítulo 7

Teoria dos Jogos

Situações onde os resultados dependem das decisões de agentes independentes são comuns em engenharia, economia e ciências humanas, entre outras. Em tais situações, podemos estar interessados em formular um problema de decisão, simular o comportamento de um conjunto de entidades competitivas ou cooperativas, e ainda buscar uma política de decisão que maximize os retornos. Com esse intuito, faremos uma breve introdução à Teoria dos Jogos, apresentando alguns conceitos básicos e formalismos, mas nos concentraremos em jogos matriciais onde o ganho de um agente invariavelmente incorre prejuízos ao outro (*finite two-person zero-sum games*).

7.1 Introdução

A Teoria dos Jogos se preocupa com procedimentos de decisão entre múltiplos agentes independentes, podendo ser de natureza *dinâmica* quando a ordem das decisões afeta o resultado final e *não-cooperativa* quando cada agente se preocupa com o benefício próprio, o que tipicamente está em conflito com os interesses dos demais agentes.

Para melhor apresentar alguns conceitos, vamos assumir que $x \in \mathbb{R}^N$ é o vetor de decisões compartilhadas por M agentes independentes, $x = (x_1 \ \ldots \ x_M)^T$, onde $x_m \in \mathbb{R}^{N_m}$ é a parte das decisões que estão sob a autoridade do m-ésimo agente e $\sum_{m=1}^{M} N_m = N$. Um jogo é dito *infinito* quando as decisões de pelo menos um agente, digamos agente m, são oriundas de um conjunto infinito de decisões, o que equivale a dizer que $x_m \in \mathbb{R}^{N_m}$. Um jogo é dito *finito* quando as decisões de todos os agentes são provenientes de conjuntos discretos, por exemplo, $x_m \in S_m \subseteq \{1, \ldots, k\}$. Um jogo é dito *dinâmico* quando as decisões dos agentes evoluem no tempo, seja de maneira contínua com $x_m(t)$ descrevendo as decisões do agente m no instante t, seja de maneira discreta com $x_m(k)$ correspondendo às decisões para o agente durante a janela de tempo $[t_k, t_{k+1})$ para $k = 0, 1, \ldots, \infty$.

É importante também destinguir entre *ações* (controles) e *estratégias* (regras de decisão ou políticas de controle). Considere um motorista que deve decidir sobre qual rota tomar para se deslocar de casa ao trabalho no dia seguinte, sendo as alternativas tomar a rota mais curta através do túnel ou seguir a rodovia ao longo do lago. Então uma estratégia para o motorista é: "se o tempo estiver chuvoso ou se o túnel estiver congestionado, então tomarei a rota mais longa ao longo do lago e, caso contrário,

tomarei o túnel". Isto é uma estratégia ou política de controle pois a decisão que será tomada depende de informações que não estão sob controle do agente. Os resultados de tal estratégia, após as informações que não podem ser antecipadas (condições climáticas e condições de tráfego) são conhecidas, são ditos ações. De certa forma, uma estratégia que não depende de condições que não podem ser antecipadas consiste de um conjunto de ações. No exemplo acima, as ações são seguir pelo túnel ou seguir pela rodovia ao longo do lago. Estratégias podem ser determinísticas, quando as ações são funções das condições observadas (estado) ou estocásticas quando as ações dependem das condições mas são tomadas probabilisticamente, podendo certas ações ter mais ou menos probabilidade do que outras ações alternativas. Há jogos onde a estratégia ótima deve ser necessariamente estocástica. Um exemplo será visto na parte de jogos matriciais na seqüência.

A noção de otimalidade também merece menção. Para tanto, seja $f_m(x)$ a função ganho que o m-ésimo agente está tentando maximizar, para $m = 1, \ldots, M$. Observe que f_m depende também das decisões de outros agentes. Surge portanto um jogo dinâmico. Seja $x(k)$ o agregado das decisões dos agentes no instante t_k. Dado $x(k)$, o melhor curso de ação do agente m é uma decisão do seu conjunto reativo $R_m(x(k))$ no ponto $x(k)$, sendo este definido por:

$$R_m(x) = \{x_m : x_m = \text{argmax}_{y_m}\ f_m(x_1, \ldots, x_{m-1}, y_m, x_{m+1}, \ldots, x_M)\}$$

O conjunto reativo $R_m(x)$ contém as soluções ótimas do problema de maximizar f_m sob a condição que os demais agentes mantêm fixas as suas decisões. Logo, o agente m implementa uma decisão $x_m(k+1) \in R_m(x(k))$ no instante t_{k+1}. Com todos os agentes reagindo às decisões dos demais com ações dos seus respectivos conjuntos reativos surge um processo iterativo:

$$x(k+1) = G(x(k)) \tag{7.1}$$

onde $G(x) = (G_1(x)\ \ldots\ G_m(x))^T$ e $G_m(x(k)) \in R_m(x(k))$ define a função reativa de cada agente $m = 1, \ldots, M$. Note que o processo iterativo (7.1) só pode ser estável quando nenhum agente tem incentivo para se desviar de suas decisões correntes, o que equivale a dizer que:

$$x(k+1) = G(x(k)) \ \Leftrightarrow\ x^* = G(x^*)$$

sendo x^* um ponto fixo. O ponto fixo x^* é um ponto estável para agentes competitivos: qualquer agente m que implementar uma solução $x_m \neq x_m^*$ estará incorrendo perdas a si próprio. Tais pontos são conhecidos por pontos *Nash*. Surge portanto a questão de convergência de processos iterativos descritos por (7.1). Não é difícil mostrar que uma condição suficiente para convergência é satisfeita quanto $\|G(x) - G(y)\| < \|x - y\|$ para alguma norma vetorial $\| \bullet \|$.

Em jogos com soma zero (*zero-sum games*), onde o ganho de um agente corresponde à perda de outro agente, não há possibilidade para ganho mútuo que permita cooperação, a menos que certos agentes sejam altruístas. Na próxima seção vamos estudar políticas de decisão que induzem equilíbrio em jogos sem possibilidade de cooperação.

Em situações onde há possibilidade de cooperação, que levam dois ou mais agentes a obterem ganhos maiores, o ponto Nash tipicamente induz ganhos inferiores. Os pontos ótimos são conhecidos por soluções Pareto. Um vetor de decisões x^P pertence

ao conjunto Pareto, denotado por $\mathcal{P}$, se não existe um vetor de decisões x tal que $f_m(x) \geqslant f_m(x^P)$ para $m = 1, \ldots, M$ e $f_m(x) > f_m(x^P)$ para algum m. Em outras palavras, $x^P \in \mathcal{P}$ se não existe um conjunto de decisões que induza ganhos melhores do que os induzidos por x^P para todos os agentes.

O restante do capítulo se concentra em jogos matriciais, que são mais simples e que podem ser resolvidos por meio da programação linear, o que consiste em encontrar os pontos de equilíbrio do jogo, ou seja, as estratégias para os jogadores que induzem um ponto Nash.

7.2 Jogos Matriciais

Em jogos matriciais cada agente seleciona uma ação dentre um número finito de possibilidades que não depende da escolha do agente oponente. O agente 1, denotado por agente linha, escolhe a ação i enquanto que o agente 2, denotado por agente coluna, escolhe a ação j. Para o par de ações (i, j), o agente 1 paga a_{ij} unidades ao agente 2 se $a_{ij} > 0$; caso contrário, se $a_{ij} < 0$, o agente 2 paga $-a_{ij}$ unidades ao agente 1. Assim o jogo matricial é representado por uma matriz $A \in \mathbb{R}^{m \times n}$ onde o agente linha pode escolher dentre m ações e o agente coluna tem n ações.

7.2.1 O Jogo da Tesoura, Pedra e Papel

Conforme matriz dada em (7.2), cada iteração do jogo tem como resultado vitória para um dos agentes (e derrota para o outro agente) ou empate. Neste jogo, nenhum agente possui uma estratégia determinística vencedora: se um agente sempre escolhe uma alternativa, então o outro agente pode selecionar a opção que o leva à vitória. Portanto, os agentes devem randomizar suas estratégias mas, em virtude da simetria do problema, cada agente deve selecionar uma das alternativas com probabilidade 1/3. Por exemplo, se o agente linha seleciona *papel* com probabilidade 1/2 e escolhe *pedra* e *tesoura* com probabilidade 1/4, então se o agente coluna selecionar *tesoura* com probabilidade 1 ele vencerá em 50% das vezes, empatará em 25% das vezes e será derrotado em apenas 25% das vezes.

$$A = \begin{array}{c} \\ Papel \\ Tesoura \\ Pedra \end{array} \begin{array}{c} \begin{array}{ccc} Papel & Tesoura & Pedra \end{array} \\ \begin{bmatrix} 0 & 1 & -1 \\ -1 & 0 & 1 \\ 1 & -1 & 0 \end{bmatrix} \end{array} \tag{7.2}$$

7.2.2 Um Jogo Menos Trivial

Considere um jogo cuja matriz de ganhos e perdas é definida como segue:

$$A = \begin{bmatrix} 0 & 1 & -2 \\ -3 & 0 & 4 \\ 5 & -6 & 0 \end{bmatrix} \tag{7.3}$$

Este jogo tem a propriedade de que qualquer estratégia fixa pode ser explorada pelo agente oponente. Por exemplo, se o agente linha escolha a primeira linha em todas as

jogadas, então o agente coluna pode escolher a segunda coluna obtendo um ganho de 1 unidade a cada iteração. Novamente, os agentes devem randomizar suas estratégias. Mas agora a probabilidade uniforme de 1/3 não é mais ótima. Além disso, qual dos agentes tem vantagem neste jogo? Note que:

$$\sum_{i,j:a_{ij}>0} a_{ij} = 10$$
$$-\sum_{i,j:a_{ij}<0} a_{ij} = 11$$

o que nos leva a suspeitar que o agente linha tem vantagem neste jogo.

7.3 Formalização

Se o agente linha segue uma estratégia randomizada, então o agente coluna observa que a probabilidade do agente linha escolher a opção i é y_i. O vetor de probabilidades $y \in \mathbb{R}^m$ do agente linha é chamado de vetor estocástico, o qual satisfaz as propriedades:

$$y \geqslant 0,\ e^T y = 1$$

onde e é um vetor com todas as entradas iguais a 1. Se a estratégia do agente coluna for modelada por um vetor estocástico x, então o ganho esperado do agente coluna é dado por:

$$\sum_{i=1}^{m}\sum_{j=1}^{n} y_i a_{ij} x_j = y^T A x.$$

7.4 Estratégia Ótima para o Agente Coluna

Suponha que o agente coluna decide jogar de acordo com a estratégia x. Nesta situação, a melhor defesa do agente linha é a estratégia y^* que resolve o seguinte problema:

$$\begin{array}{rcll} y^* & = & \underset{y \in \mathbb{R}^m}{\text{Argmin}} & y^T A x \\ & & \text{Sujeito a :} & \\ & & & e^T y = 1 \\ & & & y \geqslant 0 \end{array} \tag{7.4}$$

A partir do Teorema Fundamental da Programação Linear, sabemos que existe pelo menos uma solução básica ótima. Por exemplo, para a matriz A dada em (7.3), suponha que $x = (1/3, 1/3, 1/3)$. Então:

$$Ax = \begin{bmatrix} 0 & 1 & -2 \\ -3 & 0 & 4 \\ 5 & -6 & 0 \end{bmatrix} x = \begin{bmatrix} 0 & 1 & -2 \\ -3 & 0 & 4 \\ 5 & -6 & 0 \end{bmatrix} \begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix} = \begin{bmatrix} -1/3 \\ 1/3 \\ -1/3 \end{bmatrix}$$

Assim a melhor estratégia para o agente linha é $y' = (1, 0, 0)$, $y'' = (0, 0, 1)$, ou qualquer combinação convexa de y' e y''. Uma vez que dado x o agente linha escolhe uma

estratégia y que produz o mínimo de (7.4), então o agente coluna pode se antecipar escolhendo a estratégia x^* que produz o máximo do seguinte problema:

$$\begin{array}{rcl} x^* & = & \underset{x}{\text{Argmax}} \ \underset{y}{\text{Min}} \ y^T A x \\ & & \text{S.a :} \\ & & e^T x = 1 \\ & & e^T y = 1 \\ & & x, y \geqslant 0 \end{array} \tag{7.5}$$

Enquanto que o problema (7.4) pode ser resolvido através de programação linear, o problema (7.5) envolve dois operadores de otimização. De que forma poderíamos resolver (7.5)? O problema (7.5) pode ser resolvido por meio de programação linear. Lembre que já verificamos como que o problema interno (*Min*) pode ser resolvido por meio de uma estratégia determinística:

$$\underset{x}{\text{Max}} \ \underset{y}{\text{Min}} \ y^T A x \ = \ \underset{x}{\text{Max}} \ \text{Min} \ \{e_1^T A x, e_2^T A x, \ldots, e_m^T A x\} \tag{7.6}$$

onde e_i é um vetor com todas as entradas iguais a zero, com exceção da i-ésima entrada. O problema (7.6) pode ser expresso em programação linear como segue:

$$\begin{array}{lll} P_x : \ \text{Max} \ z & & \text{Max} \ z \\ \text{S.a :} & & \text{S.a :} \\ \quad z \leqslant e_1^T A x & \equiv & \quad ez \leqslant Ax \\ \quad \vdots & & \quad e^T x = 1 \\ \quad z \leqslant e_m^T A x & & \quad x \geqslant 0 \\ \quad e^T x = 1 & & \quad z \text{ irrestrito em sinal} \\ \quad x \geqslant 0 & & \\ \quad z \text{ irrestrito em sinal} & & \end{array}$$

O problema acima pode se colocado em uma forma matricial:

$$\begin{array}{rl} P_x : \ \text{Max} & \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ z \end{bmatrix} \\ \text{S.a :} & \\ & \begin{bmatrix} -A & e \\ e^T & 0 \end{bmatrix} \begin{bmatrix} x \\ z \end{bmatrix} \begin{matrix} \leqslant \\ = \end{matrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} \\ & x \geqslant 0 \\ & z \text{ irrestrito em sinal} \end{array} \tag{7.7}$$

Os desenvolvimentos acima mostram que a estratégia ótima x^* do agente coluna é precisamente a solução do problema P_x.

7.5 Estratégia Ótima para o Agente Linha

Por simetria, pode ser verificado que o agente linha procura uma estratégia y^* que produz a solução ótima do seguinte problema min-max:

$$\begin{array}{lllll} \underset{y}{\text{Min}} & \underset{x}{\text{Max}} & y^TAx & \equiv & \underset{y}{\text{Min}} \quad \text{Max} \quad \{y^TAe_1, y^TAe_2, \ldots, y^TAe_n\} \\ & \text{S.a :} & & & \text{S.a :} \\ & & e^Ty = 1 & & e^Ty = 1 \\ & & e^Tx = 1 & & y \geqslant 0 \\ & & x, y \geqslant 0 & & \end{array} \tag{7.8}$$

O problema (7.8) pode ser expresso em programção linear como segue:

$$\begin{array}{llll} P_y : & \text{Min} & w & \\ & \text{S.a :} & & \\ & & w \geqslant y^TAe_1 & \equiv \\ & & \vdots & \\ & & w \geqslant y^TAe_n & \\ & & e^Ty = 1 & \\ & & y \geqslant 0 & \\ & & w \text{ irrestrito em sinal} & \end{array} \qquad \begin{array}{ll} \text{Min} & w \\ \text{S.a :} & \\ & ew \geqslant A^Ty \\ & e^Ty = 1 \\ & y \geqslant 0 \\ & w \text{ irrestrito em sinal} \end{array}$$

Da mesma forma que para o agente coluna, podemos expressar o problema de encontrar uma estratégia ótima em uma forma matricial:

$$\begin{array}{lll} P_y : & \text{Min} & \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} y \\ w \end{bmatrix} \\ & \text{S.a :} & \\ & & \begin{bmatrix} -A^T & e \\ e^T & 0 \end{bmatrix} \begin{bmatrix} y \\ w \end{bmatrix} \begin{matrix} \geqslant \\ = \end{matrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} \\ & & y \geqslant 0 \\ & & w \text{ irrestrito em sinal} \end{array} \tag{7.9}$$

7.6 Relação entre os Problemas P_x e P_y

Considere o problema P_x na sua forma matricial (7.7). Vamos obter o problema dual de P_x introduzindo as variáveis duais y (para as linhas da matriz A) e a variável w para a última restrição de P_x, conforme desenvolvimento abaixo:

$$\begin{array}{lll} P_x : & \text{Max} & \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ z \end{bmatrix} \\ & \text{S.a :} & \\ & & \begin{matrix} (y) \\ (w) \end{matrix} \begin{bmatrix} -A & e \\ e^T & 0 \end{bmatrix} \begin{bmatrix} x \\ z \end{bmatrix} \begin{matrix} \leqslant \\ = \end{matrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} \\ & & x \geqslant 0 \\ & & z \text{ irrestrito.} \end{array} \tag{7.10}$$

O dual do problema (7.10) é:

$$\begin{array}{rll} \bar{P}_x: & \text{Min} & \left[\begin{array}{cc} 0 & 1 \end{array}\right] \left[\begin{array}{c} y \\ w \end{array}\right] \\ & \text{S.a:} & \\ & & \left[\begin{array}{cc} y^T & w^T \end{array}\right] \left[\begin{array}{cc} -A & e \\ e^T & 0 \end{array}\right] \begin{array}{c} \geqslant \\ = \end{array} \left[\begin{array}{c} 0 \\ 1 \end{array}\right]^T \\ & & y \geqslant 0 \\ & & w \text{ irrestrito} \end{array} \tag{7.11}$$

que pode ser manipulado de forma a se obter:

$$\begin{array}{rll} \bar{P}_x: & \text{Min} & \left[\begin{array}{cc} 0 & 1 \end{array}\right] \left[\begin{array}{c} y \\ w \end{array}\right] \\ & \text{S.a:} & \\ & & \left[\begin{array}{cc} -A^T & e \\ e^T & 0 \end{array}\right] \left[\begin{array}{c} y \\ w \end{array}\right] \begin{array}{c} \geqslant \\ = \end{array} \left[\begin{array}{c} 0 \\ 1 \end{array}\right] \\ & & y \geqslant 0 \\ & & w \text{ irrestrito} \end{array} \tag{7.12}$$

Assim concluímos que o dual de P_x corresponde ao problema P_y, ou seja, $\bar{P}_x \equiv P_y$. Ao resolver seu problema P_x, o agente coluna obtém sua estratégia ótima x^* e também a estratégia ótima y^* do agente linha.

7.7 Teorema Minimax

Inicialmente demonstrado por John Von Neumann em 1928, pode ser facilmente demonstrado através da Teoria da Dualidade Linear.

Teorema 7.1 *Existem vetores estocásticos x^* e y^* para os quais*

$$\underset{x}{Max}\ \ y^{*T}Ax \ = \ \underset{y}{Min}\ \ y^TAx^*.$$

Prova: Observando que o dual de P_x é P_y e que ambos os problemas têm regiões factíveis limitadas, concluímos que $z^* = w^*$ sendo z^* o valor ótimo de P_x e w^* o valor ótimo de P_y. Pelo lado de P_x, deduzimos que:

$$\begin{array}{rcll} z^* & = & \max\limits_x \min\{e_1^TAx, \ldots, e_m^TAy\} & \\ & = & \min\{e_1^TAx^*, \ldots, e_m^TAx^*\} & \text{[Assumindo que } x^* \text{ é solução ótima de } P_x\text{]} \\ & = & \min\limits_y y^TAx^* & \text{[Há uma solução básica ótima]} \end{array} \tag{7.13}$$

Pelo lado de P_y, deduzimos que:

$$\begin{array}{rcll} w^* & = & \min\limits_y \max\{y^TAe_1, \ldots, y^TAe_n\} & \\ & = & \max\{y^{*T}Ae_1, \ldots, y^{*T}Ae_n\} & \text{[Assumindo que } y^* \text{ é solução ótima de } P_y\text{]} \\ & = & \max\limits_x y^{*T}Ax & \text{[Há uma solução básica ótima]} \end{array} \tag{7.14}$$

A partir dos desenvolvimentos acima concluímos que:

$$\min_y y^T A x^* = z^* = w^* = \max_x y^{*T} A x$$

■

O valor ótimo $z^* = w^*$ dos problemas primal e dual é chamado de valor ótimo do jogo. Ao adotar a estratégia y^*, o agente linha garante que não perderá mais do que w^* unidades em média. Similarmente, o agente coluna assegura que receberá pelo menos z^* unidades em média se adotar a estratégia x^*.

Um jogo com valor $z^* = w^* = 0$ é dito justo. Jogos cujos papéis dos agentes podem ser invertidos são ditos jogos simétricos. Esses jogos são justo e caracterizados por $a_{ij} = -a_{ji}$ para todo i, j, isto é, $A = -A^T$. Uma matriz A é dita anti-simétrica se $A = -A^T$.

7.8 Jogos Quadráticos

Aqui vamos discutir e ilustrar as questões de convergência e otimalidade em jogos dinâmicos restritos a funções quadráticas. Dentro deste contexto, o problema de decisão do agente m é dado formalmente por:

$$P_m: \quad \underset{x_m}{\text{Minimize}} \quad \tfrac{1}{2} x^T A_m x + b_m^T x + c_m$$

onde: x_m é um vetor com as decisões do agente; $x = (x_1 \ \ldots \ x_M)^T$ é um vetor com as decisões de todos os agentes; A_m é uma matriz simétrica positiva definida; b_m é um vetor; e c_m é um escalar. Quebrando A_m em sub-matrizes e b_m em sub-vetores, P_m pode ser colocado na forma:

$$P_m: \quad \underset{x_m}{\text{Minimize}} \quad \tfrac{1}{2} \sum_{i=1}^{M} \sum_{j=1}^{M} x_i^T A_{m,i,j} x_j + \sum_{i=1}^{M} b_{m,i}^T x_i + c_m$$

onde:

$$A_m = \begin{bmatrix} A_{m,1,1} & A_{m,1,2} & \ldots & A_{m,1,M} \\ A_{m,2,1} & A_{m,2,2} & \ldots & A_{m,2,M} \\ \vdots & \vdots & \ddots & \vdots \\ A_{m,M,1} & A_{m,M,2} & \ldots & A_{m,M,M} \end{bmatrix} \text{ e } b_m = \begin{bmatrix} b_{m,1} \\ b_{m,2} \\ \vdots \\ b_{m,M} \end{bmatrix}$$

Note que o conjunto reativo do agente m, R_m, é obtido nulificando o gradiente de f_m com respeito a x_m. Ou seja, $R_m(y_m) = \{x_m : \nabla_{x_m} f_m(y_m) = 0\}$. Além disso, o processo iterativo G_m surge da seleção de um elemento de R_m. Logo, $x_m(k+1) = G_m(y_m(k))$ onde G_m é uma função tal que $G_m(y_m) \in R_m(y_m)$. De acordo com a notação estabelecida, o processo iterativo do agente m em jogos quadráticos assume a forma:

$$\begin{aligned} x_m(k+1) &= G_m(y_m(k)) && (7.15) \\ &= -[A_{m,m,m}]^{-1} [\sum_{n \neq m} A_{m,m,n} x_n(k) + b_{m,m}] && (7.16) \end{aligned}$$

onde $y_m = (x_1 \ \ldots \ x_{m-1} \ x_{m+1} \ \ldots \ x_M)^T$ é o vetor com as decisões dos agentes adversários ao agente m.

Agregando os processos iterativos (7.16) de todos os agentes, podemos expressar o processo iterativo global G como sendo a solução do sistema de equações lineares abaixo:

$$Ax(k+1) = -Bx(k) - b \tag{7.17}$$

onde A, B e b são matrizes e vetores definidos como:

$$A = \begin{bmatrix} A_{1,1,1} & 0 & & \ldots & 0 \\ 0 & A_{2,2,2} & 0 & \ldots & 0 \\ \vdots & \vdots & \vdots & \ddots & 0 \\ 0 & 0 & \ldots & 0 & A_{M,M,M} \end{bmatrix}, \ b = \begin{bmatrix} b_{1,1} \\ b_{2,2} \\ \vdots \\ b_{M,M} \end{bmatrix} \text{ e}$$

$$B = \begin{bmatrix} 0 & A_{1,1,2} & A_{1,1,3} & \ldots & A_{1,1,M} \\ A_{2,2,1} & 0 & A_{2,2,3} & \ldots & A_{2,2,M} \\ \vdots & \vdots & 0 & \ddots & \vdots \\ A_{M,M,1} & A_{M,M,2} & \ldots & A_{M,M,M-1} & 0 \end{bmatrix}$$

A solução de (7.17) leva a um processo iterativo:

$$x(k+1) = G(x(k)) = -A^{-1}[Bx(k) + b] \tag{7.18}$$

Com relação à convergência das decisões dos agentes, se ela ocorre é para um ponto Nash x^*:

$$\begin{aligned} x^* = G(x^*) &\Leftrightarrow Ax^* = Bx^* + b \\ &\Leftrightarrow x^* = A^{-1}Bx^* + A^{-1}b \\ &\Leftrightarrow x^* = (I - A^{-1}B)^{-1}A^{-1}b = (A - B)^{-1}b \end{aligned}$$

Não é difícil mostrar que se $|||A^{-1}B||| < 1$ para alguma norma-matricial $|||\cdot|||$ induzida por uma norma-vetorial $\|\cdot\|$ então G induz um processo contrativo[1] que, por sua vez, garante convergência do processo iterativo (7.18) para o ponto Nash x^*.

7.8.1 Exemplo

Objetivando exemplificar os conceitos apresentados, tomemos como exemplo um jogo entre dois agentes cujos problemas são dados abaixo:

$$P_1: \ \underset{x_1}{\text{Min}} \quad f_1 = 44.7590x_1^2 - 28.8706x_1x_2 + 10.2410x_2^2 - 150x_1 - 20x_2 + 500$$

$$P_2: \ \underset{x_2}{\text{Min}} \quad f_2 = 19.4886x_1^2 - 34.4779x_1x_2 + 25.5114x_2^2 - 120x_1 + 10x_2 + 350$$

Não é difícil resolver os sistema de equações (7.18) e verificar que o ponto Nash é único, sendo dado por $x^* = (2.0618 \ 1.1972)^T$ o que incorre um custo $f_1(x^*) = 300.4713$

[1] $\|G(y) - G(x)\| < \|y - x\|$

ao agente 1 e $f_2(x^*) = 148.8665$ ao agente 2. Pode-se ainda verificar que o jogo é convergente ao ponto Nash qualquer que seja o conjunto de decisões iniciais.

As curvas de nível das funções objetivo dos agentes, juntamente com os conjuntos reativos $R_1(x_2)$ e $R_2(x_1)$, ponto Nash e soluções Pareto podem ser visualizados na Figura 7.1. Uma trajetória de decisões também é ilustrada. Observe que existem soluções com valores melhores do que àqueles induzidos pelo ponto Nash, ou seja, as soluções Pareto produzem melhores resultados, todavia não são pontos estáveis para agentes competitivos. A propósito, $x^P = (3.8531\ 3.5534)$ é um elemento do conjunto pareto $\mathcal{P}$, o qual induz custos $f_1(x^P) = 249.50$ e $f_2(x^P) = 62.57$ que incorrem custos reduzidos a ambos os agentes se comparados aos custos produzidos pelo ponto Nash x^*.

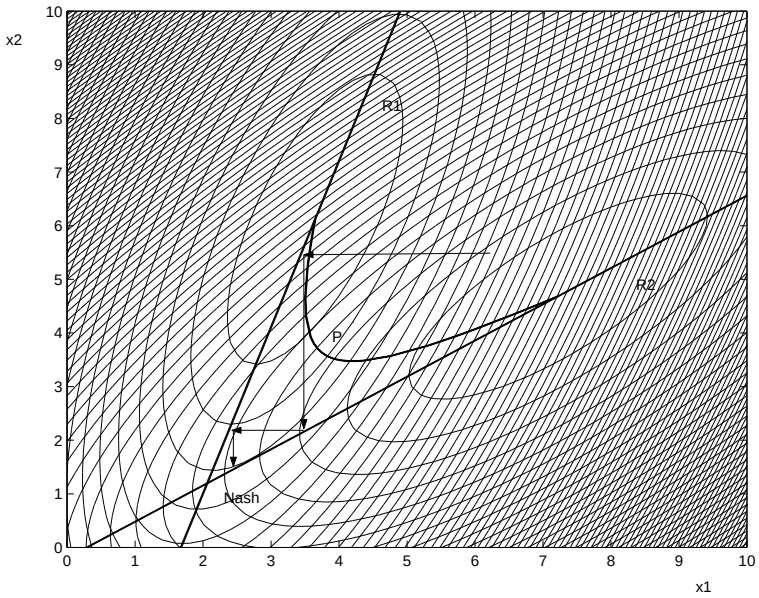


Figura 7.1: Curvas de nível, conjuntos reativos, solução Nash e pontos Pareto de um jogo quadrático entre dois agentes.

7.9 Referências

O texto de Aumann e Hart [4] é uma referência padrão para a teoria dos jogos, cobrindo a teoria, modelos e aplicações com enfoque econômico. A referência é recomendada para aqueles que desejam investigar a área mais a fundo e apreciar o espectro de aplicações. O livro de Basar e Olsder [7] também é amplamente adotado. Este texto trata de maneira rigorosa a teoria de jogos não-cooperativos, sejam eles de natureza estática ou dinâmica, discretos ou infinitos. Os autores apresentam a teoria e aplicações com enfoque em engenharia de controle que poderá ser particularmente desejável aos engenheiros de controle e elétricos. Mesterton-Gibbons [35] faz uma introdução suave e gradual à teoria dos jogos procurando integrar modelagem de fenômenos biológicos e formalismos matemáticos. Os conceitos fundamentais são apresentados por meio de

exemplos ilustrativos sobre o comportamento humano e artificial, incluindo compartilhamento de recursos, estabelecimento de preços e taxas, e competição sobre territórios.

A teoria dos jogos tem sido aplicada recentemente para facilitar o entendimento de sistemas complexos e distribuídos, bem como buscar políticas de controle que melhorem o desempenho agregado. Por exemplo, a teoria pode ser empregada para modelar problemas em robótica móvel [33] e projetar políticas de alocação de banda em redes de computadores [46].

7.10 Exercícios

EX 7.1: Primeiramente considere o jogo matricial dado pela matriz:

$$A = \begin{bmatrix} 0 & 1 & -2 \\ -3 & 0 & 4 \\ 5 & -6 & 0 \end{bmatrix}.$$

Encontre o vetor estocástico com a política de decisão do agente coluna resolvendo o problema P_x com um pacote de otimização linear tal como `lp_solve`. Encontre a solução dual de P_x diretamente a partir da solução de P_x. Repita os passos anteriores para o agente linha. O jogo dado por A é justo?

Ex 7.2: Dois jogadores, A e B, escondem independentemente uma moeda de 5 ou 10 centavos. Se as moedas escondidas possuem a mesma denominação, então o jogador A recebe ambas; se as moedas são de denominação diferente, então o jogador B recebe ambas.

i. Encontre as estratégias ótimas.

ii. Qual dos jogadores tem vantagem?

iii. Resolva o problema para quaisquer denominações a e b.

Ex 7.3: Dois jogadores, A e B, escolhem independentemente um número entre 1 e 100. Os jogadores empatam se eles escolhem o mesmo número. De outra forma, o jogador que escolheu o menor número ganha, digamos x, a menos que o oponente tenha escolhido precisamente o número $x + 1$. Encontre a estratégia ótima para os jogadores. (Sugestão: utilize um pacote de programação linear, tal como "lp_solve").

EX 7.4: Considere um jogo onde dois agentes procuram minimizar funções custo que dependem das decisões do outro. Seja $x \in \mathbb{R}^M$ o vetor de decisões. Cada agente m tem autoridade sobre a variável x_m e tem como objetivo resolver o problema:

$$P_m : \quad \underset{x_m}{\text{Minimize}} \quad f_m(y_m) = \tfrac{1}{2} x^T A_m x + b_m^T x + c_m$$

onde $y_m = [x_j : j \neq m]$. Considere um jogo onde $M = 2$,

$$A_1 = \begin{bmatrix} 89.5180 & -28.8706 \\ -28.8706 & 20.4820 \end{bmatrix},\ b_1 = \begin{bmatrix} -150 \\ 20 \end{bmatrix},\ c_1 = 0,\ \text{e}$$

$$A_2 = \begin{bmatrix} 38.9772 & -34.4779 \\ -34.4779 & 51.0228 \end{bmatrix},\ b_2 = \begin{bmatrix} -120 \\ 0 \end{bmatrix},\ c_2 = 0.$$

Se cada agente m busca iterativamente minimizar os seus custos, implementando a decisão x_m que minimiza f_m em resposta aos valores de y_m, podemos afirmar que as iterações $x(k)$ convergem para um ponto fixo x^*? Um ponto é dito fixo se as decisões dos agentes não se alteram. Tal ponto é também conhecido como ponto Nash. Se existir um ponto Nash, este ponto é um único? Podemos afirmar que o ponto Nash é um ponto Pareto (solução global ótima)?

Capítulo 8

Fluxo em Redes

Neste capítulo nos concentraremos em problemas de programação linear que apresentam uma estrutura de redes, ou seja, problemas cujas variáveis advêm de valores a serem definidos para os arcos de um grafo direcionado. Na literatura existem algoritmos específicos para resolver problemas desta classe de uma forma mais eficiente do que com algoritmos de propósito geral. O problema mais geral desta classe é o problema de fluxo em redes de custo mínimo, para o qual uma variedade de problemas práticos podem ser reduzidos e resolvidos eficientemente.

Visando motivar o estudo da otimização em redes, a Seção 8.1 apresenta os problemas clássicos de transporte e de alocação. Ambos foram estudados extensivamente na literatura e podem ser reduzidos ao problema de fluxo de custo mínimo. Na Seção 8.2 investigaremos o problema de fluxo máximo que corresponde a enviar a quantidade máxima de fluxo de um vértice fonte s a um vértice destino t, enquanto que respeitando as capacidades dos arcos. Apresentaremos algumas aplicações de fluxo máximo, o algoritmo de caminhos aumentantes e o *Teorema de Fluxo-Máximo Corte-Mínimo*. A Seção 8.3 se concentra no modelo mais geral de fluxo de custo mínimo, descrevendo aplicações, operadores de transformação entre modelos e um algoritmo básico para resolução do problema. Por fim, a Seção 8.4 faz uma introdução à propriedade de unimodularidade total de matrizes de incidência de redes e suas implicações.

8.1 Dois Problemas Clássicos

No que segue apresentamos dois problemas clássicos que podem ser reduzidos ao problema de fluxo em redes de custo mínimo.

8.1.1 O Problema de Transporte

Neste problema nos são dados um conjunto de fornecedores e um conjunto de clientes. Os fornecedores têm suas capacidades de produção limitadas, já os clientes possuem demandas a serem supridas. O custo unitário de transporte do depósito do fornecedor s_i para o cliente t_j é dado por c_{ij}. A Figura 8.1 traz uma ilustração do problema de transporte. Desejamos então encontrar a quantidade a ser produzida por cada fornecedor e as quantidades a serem enviadas aos clientes de maneira que o custo total

seja o menor possível, ao mesmo tempo que as restrições de capacidade de produção e de demanda sejam respeitadas. Os dados de uma instância do problema são:

- $S = \{s_1, \ldots, s_m\}$ é o conjunto de fornecedores;
- $T = \{t_1, \ldots, t_n\}$ é o conjunto de clientes;
- u_i é a capacidade de produção do fornecedor s_i;
- d_j é a demanda do cliente t_j; e
- c_{ij} é o custo unitário de transporte do fornecedor s_i para o cliente t_j.

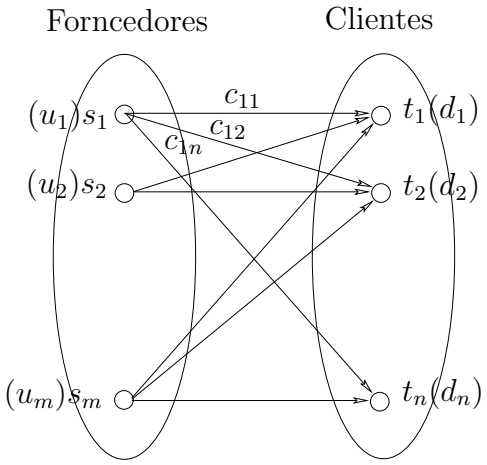


Figura 8.1: Problema de transporte.

O problema de encontrar a alocação de produção e transporte entre fornecedores e clientes que minimiza o custo agregado de transporte (problema de transporte) pode ser formulado em programação matemática:

$$
\begin{aligned}
\text{Minimize} \quad & \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij} \\
\text{Sujeito a :} \quad & \\
& \sum_{i=1}^{m} x_{ij} = d_j && j = 1, \ldots, n \\
& \sum_{j=1}^{n} x_{ij} \leq u_i && i = 1, \ldots, m \\
& x_{ij} \geq 0 && i = 1, \ldots, m \\
& && j = 1, \ldots, n
\end{aligned}
$$

onde x_{ij} é a quantidade produzida/enviada pelo fornecedor s_i ao cliente t_j. Uma instância exemplo do problema de transporte é dada nas Tabelas 8.1, 8.2 e 8.3. Observe que a capacidade total de produção supera a demanda agregada dos clientes: $\sum_{i=1}^{m} u_i = 284 > 275 = \sum_{j=1}^{n} d_j$. De outra forma o problema seria infactível. A solução ótima

para este problema tem custo 25919 sendo dada por:

$$\begin{cases} x_{11} = 0 \\ x_{13} = 39 \\ x_{14} = 87 \end{cases}, \quad \begin{cases} x_{21} = 56 \\ x_{22} = 0 \end{cases}, \text{ e } \begin{cases} x_{31} = 6 \\ x_{32} = 83 \\ x_{33} = 0 \\ x_{34} = 4 \end{cases}$$

Tabela 8.1: Capacidade de produção dos fornecedores

Fornecedores	s_1	s_2	s_3
Capacidade	135	56	93

Tabela 8.2: Demanda dos clientes

Cliente	t_1	t_2	t_3	t_4
Demanda	62	83	39	91

Tabela 8.3: Custo unitário de transporte

	Fornecedores		
Cliente	s_1	s_2	s_3
t_1	132	85	106
t_2	$+\infty$	91	89
t_3	97	$+\infty$	100
t_4	103	$+\infty$	98

8.1.2 O Problema de Alocação

O problema de alocação (*assignment problem*) consiste em designar n pessoas a n tarefas com vistas a maximizar algum critério de competência/aptidão. Seja c_{ij} o nível de competência/aptidão da pessoa i ao executar a tarefa j. Podemos então expressar o problema em programação matemática:

$$\begin{aligned} &\text{Maximize} \quad \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} \\ &\text{Sujeito a :} \\ &\qquad \sum_{i=1}^{n} x_{ij} = 1 \qquad\qquad j = 1,\ldots,n \\ &\qquad \sum_{j=1}^{n} x_{ij} = 1 \qquad\qquad i = 1,\ldots,n \\ &\qquad x_{ij} \in \{0,1\} \qquad\qquad i = 1,\ldots,n \\ &\qquad\qquad\qquad\qquad\quad j = 1,\ldots,n \end{aligned} \tag{8.1}$$

onde x_{ij} assume o valor 1 se a pessoa i é designada à tarefa j e 0 caso contrário. A primeira restrição de (8.1) garante que cada tarefa terá exatamente uma pessoa alocada. A segunda restrição força cada pessoa a ficar responsável por exatamente uma tarefa.

O poliedro definido pelas duas primeiras famílias de restrições de (8.1) e as restrições $0 \leq x_{ij} \leq 1$, $\forall i, j$, tem todos os vérticies inteiros. Isto equivale a dizer que as soluções básicas da relaxação contínua de (8.1) são inteiras e, portanto, a relaxação linear resolve o problema de alocação: basta substituir as restrições de integralidade $x_{ij} \in \{0, 1\}$ com $0 \leq x_{ij} \leq 1$ e aplicar o algoritmo Simplex. Esta propriedade advém da matriz de restrições N do problema ser totalmente unimodular, isto é, o determinante de qualquer submatriz quadrada de N é 0, 1 ou -1. Mais à frente vamos estudar esta propriedade em maiores detalhes.

8.2 O Problema de Fluxo Máximo

Dada uma rede $G = (V, A)$ com capacidade nos arcos, desejamos determinar o fluxo máximo que pode ser enviado de um vértice s (*fonte*) a um vértice t (*destino*), satisfazendo as restrições de capacidade dos arcos e as equações de equilíbrio de fluxo em todos os vértices. Em programação matemática, o problema de fluxo máximo é expresso como:

$$\begin{array}{ll} \text{Maximize} & v \\ \text{Sujeito a :} & \\ & \displaystyle\sum_{\{j:(i,j)\in A\}} x_{ij} - \sum_{\{j:(j,i)\in A\}} x_{ji} = \begin{cases} v & \text{para } i = s \\ 0 & \text{para todo } i \in V - \{s, t\} \\ -v & \text{para } i = t \end{cases} \\ & 0 \leq x_{ij} \leq u_{ij} \text{ para cada } (i, j) \in A \end{array} \tag{8.2}$$

onde u_{ij} denota a capacidade de transporte do arco (i, j). Um vetor $x = [x_{ij} : (i, j) \in A]$ satisfazendo as restrições de (8.2) é dito fluxo e o escalar v correspondente é dito valor do fluxo.

O problema de fluxo máximo surge em aplicações e também como elemento-chave de algoritmos para resolução de problemas mais complexos. Por exemplo, podemos encontrar um fluxo factível para o problema de fluxo de custo mínimo (a ser estudado na próxima seção), caso exista, resolvendo um problema de fluxo máximo. Outro exemplo é o algoritmo primal-dual para fluxo de custo mínimo que a cada iteração resolve um problema de fluxo máximo [3].

Uma propriedade do problema de fluxo máximo é o *Teorema Fluxo-Máximo Corte-Mínimo* que diz que o valor do fluxo máximo de um vértice s para t é exatamente igual a capacidade do corte s-t mínimo. Um corte $[S, V - S]$ é um corte s-t se $s \in S$ e $t \in \bar{S} = V - S$. Para um corte $[S, \bar{S}]$ denotamos por $(S, \bar{S}) = \{(i, j) \in A : i \in S, j \in \bar{S}\}$ o conjunto de arestas diretas através do corte, enquanto que $(\bar{S}, S) = \{(i, j) \in A : i \in \bar{S}, j \in S\}$ denota o conjunto das arestas reversas. A capacidade do corte $[S, \bar{S}]$ é definida como $u[S, \bar{S}] = \sum_{(i,j)\in(S,\bar{S})} u_{ij}$, ou seja, $u[S, \bar{S}]$ corresponde à capacidade total das arestas diretas que cruzam o corte.

Uma aplicação do *Teorema Fluxo-Máximo Corte-Mínimo* está na computação distribuída de módulos de um algoritmo complexo em um computador paralelo dual (com

dois processadores). Seja um algoritmo constituído de módulos que podem ser executados concorrentemente. Conhecemos de antemão o custo computacional de executarmos cada módulo i no processador 1, denotado por α_i, e o custo de ser executado no processador 2, denotado por β_i. Sabemos ainda o custo c_{ij} de comunicação entre os módulos i e j caso estes módulos sejam alocados a processadores distintos, sendo nulo o custo de comunicação caso eles sejam processados em um mesmo processador. Desejamos alocar os módulos aos processadores de forma que a soma do custo de processamento e de comunicação seja o menor possível.

Este problema pode ser modelado como um problema de corte mínimo em uma rede não-direcionada e, por sua vez, resolvido com um algoritmo para o problema de fluxo máximo. Defina um vértice s representando o processador 1, um vértice t representando o processador 2, e um vértice i para cada módulo $i = 1, \ldots, n$. Para cada módulo i, insira um arco (s, i) com capacidade $u_{si} = \beta_i$, um arco (i, t) com capacidade $u_{it} = \alpha_i$, e um arco (i, j) com capacidade $u_{ij} = c_{ij}$ se os módulos interagem. A Figura 8.2 ilustra a rede obtida para um problema cujos tempos de processamento estão na Tabela 8.4 e cujos custos de comunicação são dados na Tabela 8.5.

Tabela 8.4: Custo de processamento

i	1	2	3	4	5
α_i	3	2	6	2	7
β_i	5	5	6	4	2

Tabela 8.5: Custo de comunicação

	1	2	3	4	5
1		5		1	
2	5		1		
3		1		1	10
4	1		1		1
5			10	1	

Observe que existe uma correspondência de 1-para-1 entre corte s-t na rede e alocação de módulos a processadores, sendo a capacidade do corte precisamente o custo da alocação. Seja $A_1 \cup A_2 = \{1, \ldots, n\}$, $A_1 \cap A_2 = \emptyset$, uma alocação de módulos aos processores 1 e 2, respectivamente. O custo desta alocação é dado por:

$$\sum_{i \in A_1} \alpha_i + \sum_{i \in A_2} \beta_i + \sum_{(i,j) \in A_1 \times A_2} c_{ij} \tag{8.3}$$

O corte s-t correspondente a esta alocação é $(S, \bar{S})$ com $S = \{s\} \cup A_1$ e $\bar{S} = \{t\} \cup A_2$. Este corte contém um arco (i, t) para cada $i \in A_1$ cuja capacidade é $u_{it} = \alpha_i$, um arco (s, i) para cada $i \in A_2$ cuja capacidade é $u_{si} = \beta_i$, e um arco (i, j) de capacidade $u_{ij} = c_{ij}$ para cada $(i, j) \in A_1 \times A_2$. Portanto $u[S, \bar{S}] = \sum_{i \in A_1} \alpha_i + \sum_{i \in A_2} \beta_i + \sum_{(i,j) \in A_1 \times A_2} c_{ij}$. Encontrando o corte de capacidade mínima estaremos produzindo uma alocação de

custo mínimo. Para o exemplo da Figura 8.2, o corte mínimo está indicado e este induz uma alocação $A_1 = \{1, 2, 4\}$ e $A_2 = \{3, 5\}$, tendo custo de processamento total igual a $7 + 8 + 3 = 18$, que é precisamente a capacidade do corte s-t mínimo.

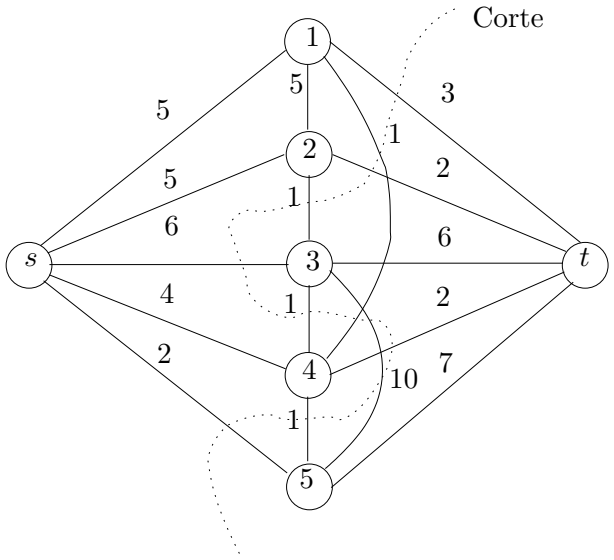


Figura 8.2: Alocação balanceada de tarefas em dois processadores como um problema de corte mínimo. A alocação ótima consiste em processar os módulos 1, 2 e 4 no processador 1, deixando os módulos 3 e 5 no processador 2. O custo de processamento dos módulos é igual 15 enquanto que o custo de comunicação é 3, perfazendo um custo total de 18 unidades. O custo total é a capacidade do corte s-t mínimo como indicado na figura.

8.2.1 Fluxos e Cortes

O conceito de rede residual é chave no desenvolvimento de algoritmos para o problema de fluxo máximo[1]. Dado um fluxo x para uma rede G, a capacidade residual r_{ij} de um arco $(i, j) \in A$ é a quantidade máxima de fluxo adicional que pode ser enviada usando os arcos (i, j) e (j, i). Vamos assumir por simplicidade que o arco (j, i) será um elemento da rede sempre que o arco (i, j) fizer parte da rede. A capacidade residual tem dois componentes: o componente $u_{ij} - x_{ij}$ (quantidade não utilizada do arco (i, j)) e o componente x_{ji} (podemos cancelar o envio de x_{ji} unidades enviadas através do arco (j, i)). Logo, $r_{ij} = u_{ij} - x_{ij} + x_{ji}$. Denotamos por $G(x)$ a *rede residual* que contém apenas arcos com capacidade residual positiva. A Figura 8.3 traz um exemplo de rede residual.

Lembramos que um corte s-t $[S, \bar{S}]$ tem um conjunto de arcos diretos representado por $(S, \bar{S})$ e um conjunto de arcos reversos denotado por $(\bar{S}, S)$. A capacidade do corte é denotada por $u[S, \bar{S}]$ sendo igual a soma da capacidade dos arcos diretos que

[1] A noção de rede residual será estendida na próxima seção para resolver o problema de fluxo de custo mínimo.

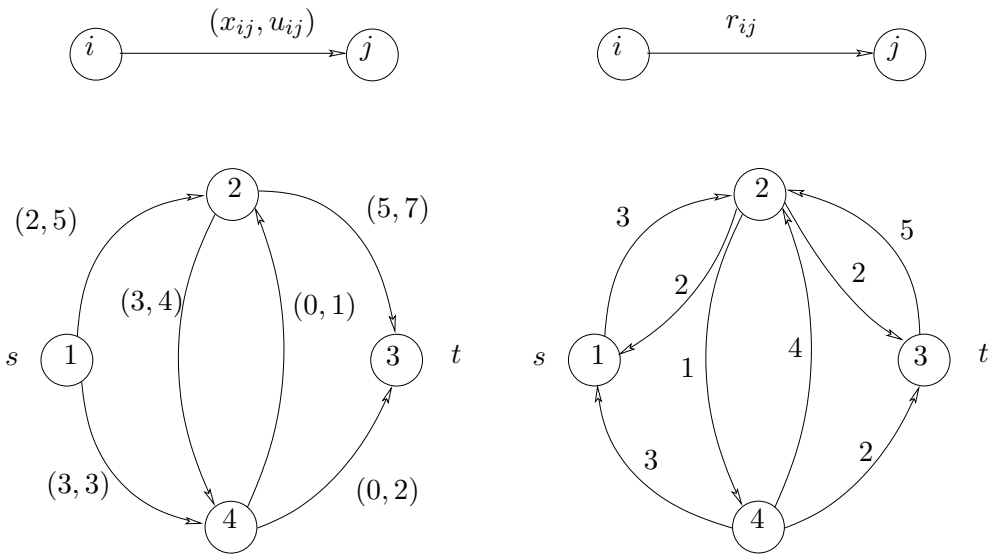


Figura 8.3: Exemplo de rede residual $G(x)$ obtida a partir de uma rede G e fluxo x.

atravessam o corte, ou seja:

$$u[S, \bar{S}] = \sum_{(i,j)\in(S,\bar{S})} u_{ij}$$

Obviamente, a capacidade de um corte s-t define um limite superior para a quantidade de fluxo que pode ser enviada de s para t. O corte cuja capacidade é mínima dentre todos os cortes s-t é dito *corte mínimo*.

A capacidade residual de um corte s-t é denotada por $r[S, \bar{S}]$ e definida como a soma das capacidades residuais dos arcos diretos do corte, isto é:

$$r[S, \bar{S}] = \sum_{(i,j)\in(S,\bar{S})} r_{ij}$$

Seja x um fluxo na rede. Adicionando as equações de conservação de fluxo do modelo (8.2) para os nós em S, obtemos:

$$v = \sum_{i\in S} \left[\sum_{\{j:(i,j)\in A\}} x_{ij} - \sum_{\{j:(j,i)\in A\}} x_{ji} \right] \tag{8.4}$$

Quando dois vértices $u, v \in S$, então x_{uv} aparece na equação de equilíbrio de u e $-x_{uv}$ aparece na equação de equilíbrio de v, dessa forma cancelando um ao outro em (8.4). O mesmo ocorre se $u, v \in \bar{S}$. Isto nos leva a concluir que a expressão (8.4) pode ser simplificada:

$$v = \sum_{(i,j)\in(S,\bar{S})} x_{ij} - \sum_{(j,i)\in(\bar{S},S)} x_{ji} \tag{8.5}$$

A primeira expressão no lado direito de (8.5) corresponde ao fluxo dos nós de S para os nós de $\bar{S}$, enquanto que a segunda nos dá o fluxo dos nós de $\bar{S}$ para S. Logo, o lado direito de (8.5) é o fluxo líquido através do corte s-t $[S, \bar{S}]$ que é precisamente

v. Substituindo $x_{ij} \leq u_{ij}$ na primeira expressão e $x_{ij} \geq 0$ na segunda, obtemos a desigualdade:

$$v \leq \sum_{(i,j)\in(S,\bar{S})} u_{ij} = u[S,\bar{S}] \tag{8.6}$$

A expressão significa que o valor de qualquer fluxo x é menor ou igual a capacidade de qualquer corte s-t. Qualquer fluxo deve atravessar todos os cortes s-t, portanto, seu valor não pode exceder a capacidade de qualquer um deles. Isto nos leva a deduzir que:

$$v \leq \min\{u[S,\bar{S}] : [S,\bar{S}] \text{ é um corte } s\text{-}t \text{ de } G\} \tag{8.7}$$

A expressão (8.7) nos diz que se encontrarmos um fluxo x cujo valor v iguala a capacidade de algum corte s-t, então x é um fluxo máximo. Tal propriedade pode ser definida em termos das capacidades residuais. Suponha que x é um fluxo e seu valor é v. Seja x' um fluxo de valor $v + \Delta v$ onde $\Delta v \geq 0$. Neste caso, a desigualdade (8.6) implica:

$$v + \Delta v \leq \sum_{(i,j)\in(S,\bar{S})} u_{ij} \tag{8.8}$$

Substituindo (8.5) em (8.8), obtemos:

$$\begin{aligned}
\Delta v &\leq \sum_{(i,j)\in(S,\bar{S})} u_{ij} - \sum_{(i,j)\in(S,\bar{S})} x_{ij} + \sum_{(j,i)\in(\bar{S},S)} x_{ji} \\
&= \sum_{(i,j)\in(S,\bar{S})} (u_{ij} - x_{ij} + x_{ji}) \qquad [\text{Assumimos que } (i,j) \in A \Leftrightarrow (j,i) \in A] \\
&= \sum_{(i,j)\in(S,\bar{S})} r_{ij} \\
&= r[S,\bar{S}]
\end{aligned} \tag{8.9}$$

Assim, concluímos que para qualquer fluxo x de valor v em uma rede G, a quantidade adicional de fluxo que pode ser enviada de um nó s para um nó t é menor ou igual a capacidade residual de qualquer corte s-t.

8.2.2 Algoritmo de Caminhos Aumentantes

Um caminho p de s para t na rede residual $G(x)$ é dito *caminho aumentante*. A capacidade do caminho aumentante é denotada por $\delta(p)$ e definida como a capacidade residual do arco (i,j) de menor capacidade residual em p. Matematicamente, $\delta(p) = \min\{r_{ij} : (i,j) \in p\}$. Considere a rede residual $G(x^0)$ para um fluxo x^0 nulo dada na Figura 8.4. O caminho $p_0 = (1,2,4,5)$ é um caminho aumentante com capacidade $\delta(p_0) = \min\{r_{12}, r_{24}, r_{45}\} = \min\{4,3,5\} = 3$. Por definição, um caminho aumentante tem capacidade não nula[2]. Logo, se $G(x)$ possui um caminho aumentante podemos enviar fluxo adicional de s para t e, portanto, x não pode ser um fluxo máximo. O algoritmo de caminhos aumentantes iterativamente busca um caminho aumentante p e envia a quantidade de fluxo correspondente a $\delta(p)$ até que não haja caminho aumentante. O pseudo-código do algoritmo segue abaixo.

[2]Arcos da rede residual têm capacidade residual não nula.

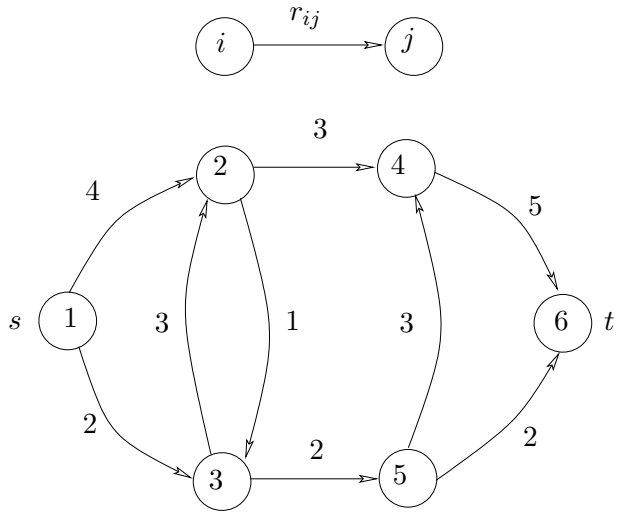


Figura 8.4: Rede residual $G(x^0)$ para fluxo x^0 nulo.

Algoritmo: Caminhos Aumentantes

$x = 0$
Enquanto $G(x)$ contém caminho aumentante faça
    Encontre um caminho p de s para t em $G(x)$
    $\delta(p) = \min\{r_{ij} : (i,j) \in p\}$
    Envie a quantidade $\delta(p)$ de fluxo ao longo de p,
        atualize x e obtenha a rede residual $G(x)$
Fim-enquanto

O comportamento do algoritmo pode ser ilustrado com a rede residual da Figura 8.4. Enviando a quantidade $\delta(p_0) = 3$ de fluxo ao longo de p_0, obtemos o fluxo x^1 com rede residual $G(x^1)$ dada na Figura 8.5.

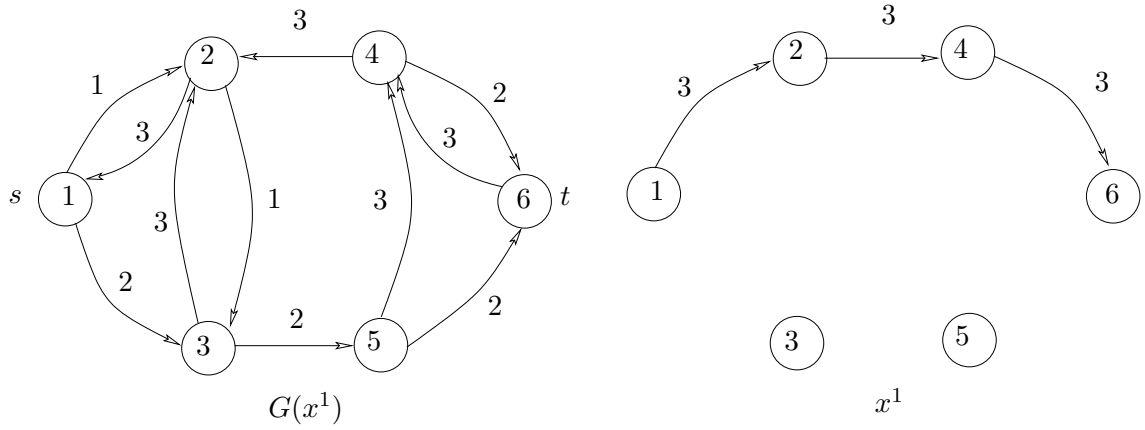


Figura 8.5: Rede residual $G(x^1)$ para fluxo x^1.

Para a rede residual $G(x^1)$, o caminho $p_1 = (1, 2, 3, 5, 4, 6)$ é um caminho aumentante com capacidade $\delta(p_1) = \min\{1, 1, 2, 3, 2\} = 1$. Enviando a quantidade $\delta(p_1)$ ao longo de p obtemos o fluxo x^2 e a rede residual $G(x^2)$ conforme Figura 8.6.

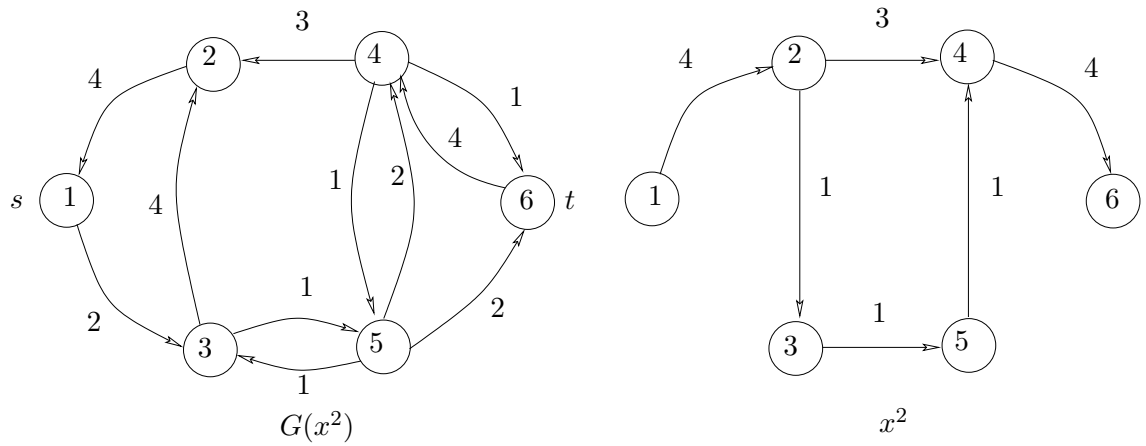


Figura 8.6: Rede residual $G(x^2)$ para fluxo x^2.

O caminho $p_2 = (1, 3, 5, 6)$ é aumentante em $G(x^2)$ e sua capacidade residual é $\delta(p_2) = \min\{2, 1, 2\} = 1$. Enviando $\delta(p_2)$ unidades através de p_2 obtemos o fluxo x^3 e a rede residual $G(x^3)$ dada na Figura 8.7.

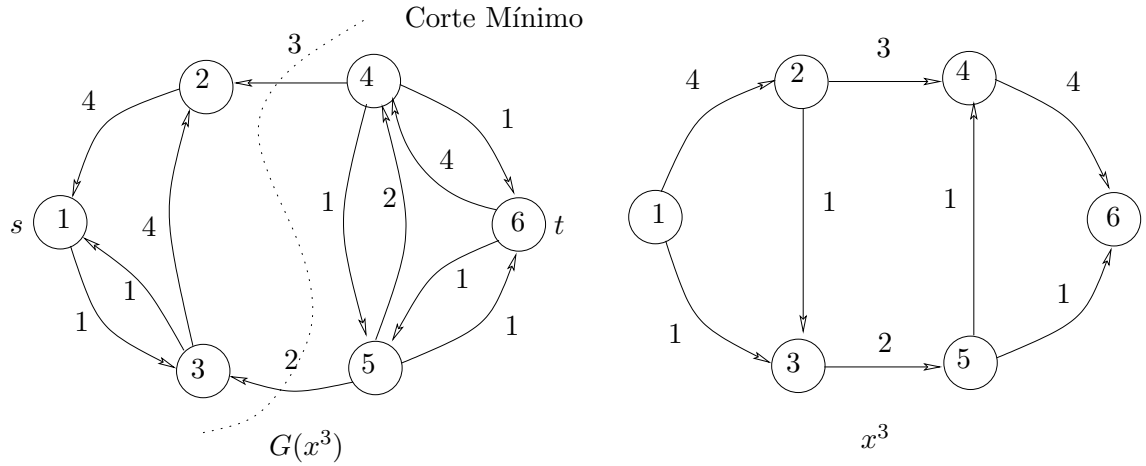


Figura 8.7: Rede residual $G(x^3)$ para fluxo x^3.

A rede residual $G(x^3)$ não possui caminho aumentante, acarretando a finalização do algoritmo. Observe que os vértices alcançáveis a partir de s em $G(x^3)$ são os elementos do conjunto $S = \{1, 2, 3\}$. Note que $[S, \bar{S}]$ forma uma corte s-t em G tal que a capacidade $u[S, \bar{S}] = 5$ é precisamente o valor v^3 do fluxo enviado de s para t segundo o fluxo x^3: p_0 enviou 3 unidades, p_1 enviou 1 unidade e p_2 enviou 1 unidade, perfazendo um total de 5 unidades enviadas. Portanto $[S, \bar{S}]$ é um corte mínimo e o fluxo x^3 é um fluxo máximo: $u[S, \bar{S}] = v^3$.

Teorema 8.1 *(*Caminho Aumentante*) Um fluxo x^* é máximo se e somente se a rede residual $G(x^*)$ não contém caminho aumentante.*

Prova: Se a rede residual $G(x^*)$ contém um caminho aumentante, então obviamente x^* não pode ser máximo. Por outro lado, se a rede residual não contém caminho

aumentante, então o conjunto de vértices alcançáveis a partir de s, denotado por S, define um corte s-t $[S, \bar{S}]$ tal que:

$$\begin{aligned} 0 &= r[S, \bar{S}] \\ &= \sum_{(i,j)\in(S,\bar{S})} r_{ij} \\ &= \sum_{(i,j)\in(S,\bar{S})} (u_{ij} - x_{ij} + x_{ji}) \end{aligned}$$

e, portanto,

$$\begin{aligned} u[S, \bar{S}] &= \sum_{(i,j)\in(S,\bar{S})} u_{ij} \\ &= \sum_{(i,j)\in(S,\bar{S})} (x_{ij} - x_{ji}) \\ &= \sum_{(i,j)\in(S,\bar{S})} x_{ij} - \sum_{(j,i)\in(\bar{S},S)} x_{ji} \\ &= v \end{aligned}$$

Concluímos que a capacidade $u[S, \bar{S}]$ do corte $[S, \bar{S}]$ iguala o valor do fluxo induzido por x^*, dessa forma demonstrando que o fluxo é máximo. ■

Qual é o tempo de execução do algoritmo de caminhos aumentantes? Podemos estabelecer um limite superior para o número de iterações do algoritmo. Seja $u = \max\{u_{ij} : (i,j) \in A\}$ a capacidade máxima dos arcos, assumindo que as capacidades são finitas e inteiras. Então o fluxo máximo tem como limite superior o valor nu com $n = |V|$ pois o corte $[\{s\}, V - \{s\}]$ tem capacidade máxima $(n-1)u$. Uma vez que todo caminho aumentante p tem capacidade não nula e inteira, concluímos que $\delta(p) \geq 1$ e o número máximo de iterações do algoritmo de caminhos aumentantes é nu. Uma busca em grafos (largura ou profundidade) pode ser realizada em tempo $O(m+n)$, o que nos leva a deduzir que o tempo de execução do algoritmo é limitado por $O(mnu)$.

8.2.3 Implicações Combinatórias do Problema de Fluxo Máximo

Alguns resultados com implicações combinatórias surgem a partir do problema de fluxo máximo. O *Teorema do Fluxo-Máximo Custo-Mínimo* mostra que o problema de encontrar um corte mínimo pode ser reduzido ao problema de fluxo máximo[3].

Teorema 8.2 *(Integralidade de Fluxo) Se as capacidades de todos os arcos são inteiros, então o problema de fluxo máximo tem valor inteiro.*

Prova: Por indução no número de caminhos aumentantes podemos demonstrar o resultado. Na iteração 0, quando o fluxo inicial x^0 é nulo, as capacidades residuais dos arcos de $G(x^0)$ são inteiras e o valor v^0 do fluxo é nulo e portanto inteiro.

[3]Observe que existe um número combinatório de cortes s-t. Qual é o número de cortes s-t?

Considere a iteração $k \geq 1$ do algoritmo de caminhos aumentantes e suponha, por indução, que as capacidades dos arcos de $G(x^k)$ são inteiras e o valor do fluxo v^k é inteiro. Então a capacidade $\delta(p_k)$ do caminho aumentante p_k em $G(x^k)$ é inteira. Enviando a quantidade $\delta(p_k)$ ao longo de p_k obtemos um fluxo x^{k+1} com quantidades inteiras, uma rede residual $G(x^{k+1})$ com capacidades inteiras e um valor $v^{k+1} = v^k + \delta(p_k)$ também inteiro. ■

Teorema 8.3 *(Fluxo-máximo corte-mínimo) O valor do fluxo máximo v que pode ser enviado de um vértice s para um vértice t iguala a capacidade $u[S, \bar{S}]$ do corte s-t de menor capacidade.*

Prova: Seja x o fluxo máximo obtido com o algoritmo de caminhos aumentantes e v o seu valor. Seja ainda S o conjunto dos vértices alcançáveis em $G(x)$ a partir de s. Segundo o Teorema 8.1, o corte $[S, \bar{S}]$ tem capacidade $u[S, \bar{S}] = v$. Uma vez que $v \leq \min\{u[U, \bar{U}] : [U, \bar{U}]$ é um corte s-$t\}$, concluímos que $v = \min\{u[U, \bar{U}] : [U, \bar{U}]$ é um corte s-$t\}$. ■

Tomando como base os resultados acima estabelecidos, podemos deduzir propriedades acerca da conexidade de vértices em grafos. Dados um grafo direcionado $G = (V, A)$ e dois vértices distintos s e t, desejamos determinar o menor número mínimo de arcos que devem ser removidos de maneira a interromper todos os caminhos de s para t. Este número é conhecido como conexidade-arco e será denotado aqui por $\alpha_{s,t}$. Para o caso direcionado, observe que $\alpha_{s,t}$ não é necessariamente igual a $\alpha_{t,s}$. Podemos nos fazer valer dos Teoremas 8.2 e 8.3 para calcular a conexidade-arco. Inicialmente, note que $\alpha_{s,t}$ é maior ou igual ao número máximo de caminhos arco-disjuntos de s para t. (É necessário remover pelo menos um arco de cada um desses caminhos para desconectar s e t.) Podemos calcular o número máximo de caminhos arco-disjuntos definindo a capacidade $u_{ij} = 1$ para todo $(i, j) \in A$ e calculando o fluxo máximo de s para t. Pelo teorema da integralidade de fluxo, o fluxo máximo de s para t corresponde a um conjunto de caminhos arco-disjuntos. Pelo teorema do fluxo-máximo corte-mínimo, $\alpha_{s,t}$ iguala a capacidade $u[S, \bar{S}]$ de um corte s-t mínimo $[S, \bar{S}]$. Daí concluímos que o número máximo de caminhos disjuntos de s para t é exatamente o número mínimo de arcos que devem ser removidos de maneira a desconectar s e t, que é precisamente a conexidade-arco $\alpha_{s,t}$.

Outra propriedade de interesse é a conexidade-vértice entre dois vértices distintos s e t. Mas antes de discutirmos esta propriedade, vamos definir a propriedade quebra de vértices sobre um grafo $G = (V, A)$, a qual produz um grafo $G' = (V', A')$ como segue: $V' = \{i, i' : i \in V\}$ e $A' = \{(i, i') : i \in V\} \cup \{(i', j) : (i, j) \in A\}$. A conexidade-vértice entre s e t é definida como o menor número de vértices, distintos de s e t, que devem ser removidos de forma a quebrar todos os caminho de s para t. Vamos assumir que $(s, t) \notin A$. Seja $\beta_{s,t}$ a conexidade-vértice entre s e t. Note que $\beta_{s,t}$ é maior ou igual ao número máximo de caminhos vértice-disjuntos de s para t. Para calcular $\beta_{s,t}$, tomemos o grafo G' obtido a partir da quebra de vértices de G. Definindo $u_{i,i'} = 1$ para todo $i \in V$ e $u_{i',j} = \infty$ para todo $(i, j) \in A$, o fluxo máximo de s' para t em G' nos dá, pelo Teorema 8.2, o conjunto máximo de caminhos vértice-disjuntos de s para t. Pelo Teorema 8.3, deduzimos que a conexidade-vértice $\beta_{s,t}$ iguala o número máximo de caminhos vértice-disjuntos que, por conseguinte, iguala a capacidade de um corte s-t

mínimo $[S, \bar{S}]$. Em outras palavras, o número máximo de camimhos vértice-disjuntos é igual ao número mínimo de vértices que devem ser removidos de maneira de desconectar s de t[4].

8.3 O Problema de Fluxo de Custo Mínimo

Um universo amplo de problemas teóricos e práticos podem ser transformados em problemas de fluxo de custo mínimo. Os problemas apresentados acima são dois exemplos. Os dados que formam uma instância do problema de fluxo de custo mínimo são:

- um grafo direcionado $G = (V, A)$ consistindo em um conjunto de vértices (ou nós) e um conjunto de arcos;
- o custo unitário c_{ij} de transporte através do arco (i, j);
- o limite inferior l_{ij} e superior u_{ij} para fluxo através do arco (i, j); e
- a quantidade de fluxo b_i que deve ser injetada ou consumida pelo nó i; se $b_i > 0$, então i é um nó fornecedor; se $b_i < 0$, então i é um nó consumidor; e se $b_i = 0$, então i é um nó de transporte.

Em programação matemática, o problema de fluxo de custo mínimo é expresso como:

$$\begin{aligned} &\text{Minimize} && \sum_{(i,j)\in A} c_{ij}x_{ij} \\ &\text{Sujeito a :} \\ &&& \sum_{\{j:(i,j)\in A\}} x_{ij} - \sum_{\{j:(j,i)\in A\}} x_{ji} = b_i, && \forall\ i \in V \\ &&& l_{ij} \le x_{ij} \le u_{ij}, && \forall\ (i,j) \in A \end{aligned} \tag{8.10}$$

A primeira família de restrições de (8.10) é conhecida por *equação de conservação de fluxo*. Em síntese, a igualdade impõe a restrição de que a quantidade de fluxo que sai do nó i menos a quantidade que entra deve ser igual a b_i. Quando $b_i > 0$, o fluxo de saída líquido deve ser igual a esta quantidade. Quando $b_i < 0$, o fluxo de entrada líquido deve ser igual a $-b_i$. Por fim, quando $b_i = 0$, a quantidade que entra deve ser precisamente igual à quantidade que sai do nó i.

As seguintes classes de problemas podem ser convertidas em problemas de fluxo de custo mínimo:

1) **Fluxo máximo em grafos:** neste problema é dado um grafo direcionado $G = (V, A)$, um nó fonte s, um nó destino t e capacidades de transporte u_{ij} para os arcos de G; deseja-se encontrar fluxos para os arcos tal que a quantidade de fluxo enviada de s para t seja maximizada, mas sem violar as capacidades dos arcos e sem acumulação de fluxo nos vértices do conjunto $V - \{s, t\}$.

[4]Note que somente arcos do tipo (i, i') podem aparecer no corte mínimo $[S, \bar{S}]$, portanto o corte mínimo de arcos em G' corresponde a um corte mínimo de vértices em G.

2) **Caminhos mínimos em grafos:** é dado um grafo direcionado $G = (V, A)$, pesos c_{ij} para os arcos, um vértice fonte s e um vértice destino t; busca-se um caminho de s para t tal que a soma dos pesos das arestas do caminho seja a menor possível.

3) **Problema de alocação:** conforme visto na Seção 8.1.

4) **Problema de transporte:** conforme discutido na Seção 8.1.

5) **Fluxo em redes com função custo convexa:** este problema é uma variação do problema de fluxo de custo mínimo onde a função custo de transporte $c_{ij}(x_{ij})$ através de cada arco (i, j) é convexa, mas não necessariamente linear e igual a $c_{ij}x_{ij}$ como na versão padrão do problema; podemos, entretanto, resolver o problema com custo convexo de forma abitrariamente aproximada linearizando por partes a função custo.

8.3.1 Transformações

Há um conjunto de transformações que tipicamente são úteis quando se trabalha com problemas de fluxo de custo mínimo. Por exemplo, é possível converter um problema de fluxo de custo mínimo cujos arcos são não direcionados em outro problema de fluxo de custo mínimo cujos arcos são direcionados. Podemos também eliminar o limite inferior l_{ij} dos arcos sem maiores dificuldades. Logo, se você tem que resolver um problema onde certos arcos têm limites inferiores, mas você dispõe de um algoritmo que só aceita limites superiores, então é possível reduzir o problema a outro problema equivalente aceito pelo algoritmo. Abaixo apresentamos algumas transformações que serão exemplificadas por meio de figuras.

Convertendo Arestas em Arcos

Considere o arco (i, j) da rede ilustrada na Figura 8.8. Assumindo que $l_{ij} = 0$ e $c_{ij} \geq 0$, é fácil verificar que uma solução ótima do problema de fluxo de custo mínimo terá necessariamente fluxo em uma direção ou outra, mas não em ambas[5]. Assim podemos transformar a aresta não direcionada em dois arcos com sentidos opostos, cada um deles com o custo e capacidade da aresta.

Seja $\bar{x}_{ij}$ o fluxo no arco não direcionado, $\bar{c}_{ij}$ o custo unitário de transporte em qualquer direção, e $\bar{u}_{ij}$ a capacidade de transporte. Transformamos este arco não direcionado $\{i, j\}$ em dois arcos direcionados, um arco (i, j) e um arco (j, i). O fluxo através de (i, j) é denotado por x_{ij}, enquanto o custo unitário fica $c_{ij} = \bar{c}_{ij}$ e a capacidade $u_{ij} = \bar{u}_{ij}$. O fluxo através de (j, i) é denotado por x_{ji}, sendo o custo unitário $c_{ji} = \bar{c}_{ij}$ e a capacidade $u_{ji} = \bar{u}_{ij}$.

Para estabelecermos a equivalência, seja $\bar{x}_{ij} = \delta$ um fluxo no arco não-direcionado em uma solução ótima na rede original. Podemos assumir que o fluxo de δ unidades é na direção (i, j) ou (j, i). Sem perda de generalidade, suponha que δ unidades são

[5] A transformação não é válida se $c_{ij} < 0$ ou $l_{ij} > 0$: se $l_{ij} > 0$ então não sabemos em que direção o fluxo deverá satisfazer esta restrição; se $c_{ij} < 0$ então não podemos garantir que o fluxo ótimo será em apenas uma direção.

enviadas de i para j. Fazendo $x_{ij} = \delta$ e $x_{ji} = 0$ na rede modificada, obtemos um fluxo equivalente e com mesmo custo. Seja agora um fluxo ótimo na rede modificada com valores x_{ij} e x_{ji}. Podemos assumir que o fluxo através de (i, j) ou através de (j, i) é nulo. Sem perda de generalidade, suponha que $x_{ij} = \delta$ enquanto que $x_{ji} = 0$. Podemos então obter um fluxo equivalente na rede original fazendo $\bar{x}_{ij}$ assumir o valor δ e na direção de i para j. Ambos os fluxo induzem a mesma parcela de custo na função objetivo. Portanto, mostramos que a tranformação produz um problema equivalente.

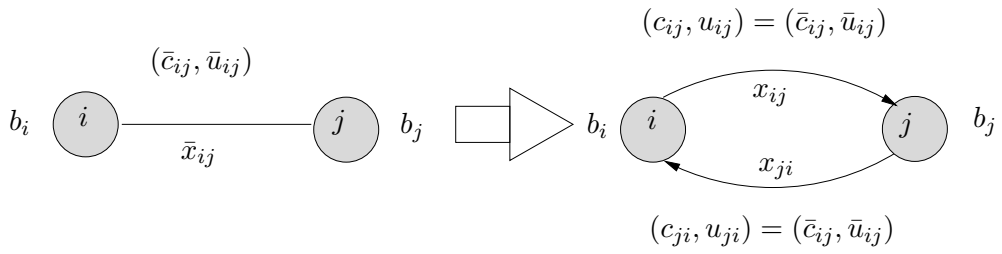


Figura 8.8: Transformação de arestas em arcos.

Removendo Limites Inferiores

Para eliminarmos o limite inferior de um arco (i, j) basta pré-enviar a quantidade l_{ij} através do arco, reduzir o suprimento do nó i de l_{ij} unidades ($b_i \to b_i - l_{ij}$), aumentar em l_{ij} unidades a quantidade de fluxo que deve emanar do nó j ($b_j \to b_j + l_{ij}$), e finalmente reduzir a capacidade do arco (i, j) em l_{ij} unidades. Note que o custo final deve ser acrescido de uma constante $c_{ij}l_{ij}$ de forma a refletir o envio antecipado do fluxo.

A remoção de limite inferior é ilustrada na Figura 8.9. Neste exemplo, o fluxo na rede original é denotado por $\bar{x}_{ij}$, onde o custo de transporte é $\bar{c}_{ij}$, o limite inferior é $\bar{l}_{ij}$ e a capacidade dada por $\bar{u}_{ij}$. As acumulações de fluxo na rede original são denotadas por $\bar{b}_i$ e $\bar{b}_j$. Então o fluxo na rede modificada passa a ser denotado por x_{ij}, com capacidade $u_{ij} = \bar{u}_{ij} - \bar{l}_{ij}$, custo unitário $c_{ij} = \bar{c}_{ij}$, e acumulações de fluxo $b_i = \bar{b}_i - \bar{l}_{ij}$ e $b_j = \bar{b}_j + \bar{l}_{ij}$. O fluxo na rede original, $\bar{x}_{ij}$, e o fluxo na rede modificada, x_{ij}, obedecem a relação $\bar{x}_{ij} = x_{ij} + \bar{l}_{ij}$.

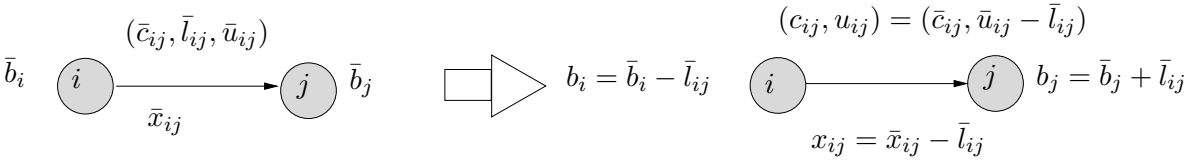


Figura 8.9: Remoção de limite inferior nos arcos.

Removendo Capacidade de Arcos

A Figura 8.10 ilustra como que um arco com capacidade pode ser substituído por dois arcos sem capacidade por meio da inserção de um nó auxiliar. É fácil verificar que o fluxo x_{ik} através do arco (i, k) da rede modificada corresponde precisamente ao fluxo $\bar{x}_{ij}$ através do arco (i, j) da rede original. Note que os fluxos satisfazem a relação

$x_{ik} + x_{jk} = \bar{u}_{ij}$, portanto x_{jk} é o resíduo de capacidade de transmissão através do arco (i, j) (*slack*).

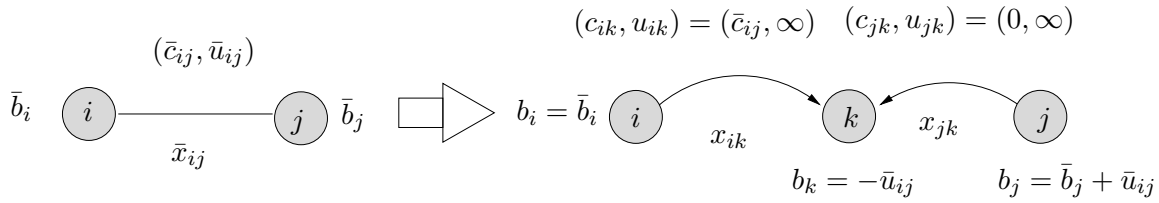


Figura 8.10: Eliminando capacidade de arcos.

8.3.2 Um Exemplo

Na Figura 8.11 é ilustrado um problema de fluxo de custo mínimo, consistindo em enviar 10 unidades de fluxo do nó 1 (produtor) para o nó 4 (consumidor). Os arcos sólidos indicam os arcos da rede, enquanto que os arcos tracejados indicam a solução ótima do problema.

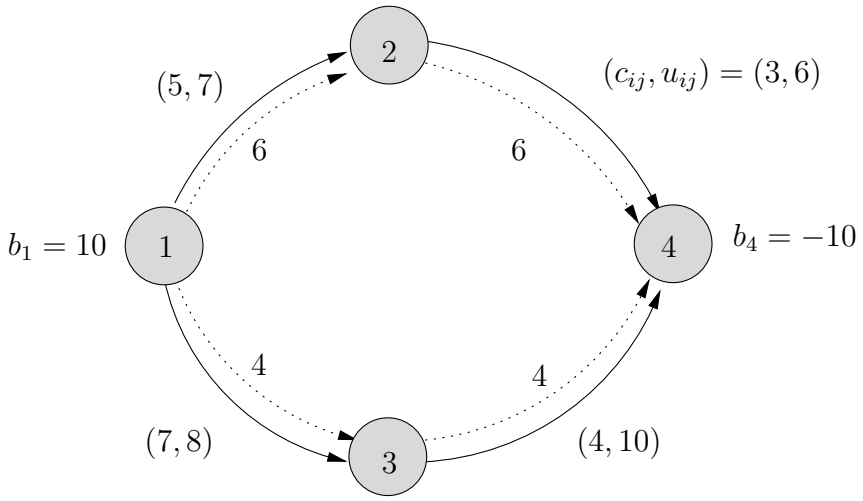


Figura 8.11: Exemplo de problema de fluxo de custo mínimo. As linhas tracejadas correspondem ao fluxo ótimo.

Em geral, o problema de fluxo de custo mínimo especificado na forma (8.10) pode ser colocado de forma mais compacta como:

$$\begin{array}{ll} \text{Min} & c^T x \\ \text{S.a :} & Nx = b \\ & l \leq x \leq u \\ & x \in \mathbb{R}^{|A|} \end{array}$$

onde c é vetor de custo unitário de transporte, N é a matriz de incidência do grafo, b é o vetor com as taxas de injeção e consumo de fluxo, l é o vetor com limites inferiores, u é o vetor de limites superiores, e x é o vetor com o fluxo dos arcos. Para o problema

dado na Figura 8.11, os dados para uma formulação compacta são:

$$N = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} (1,2) & (1,3) & (2,4) & (3,4) \end{array} \\ \begin{bmatrix} 1 & 1 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 0 & -1 & 0 & 1 \\ 0 & 0 & -1 & -1 \end{bmatrix} \end{array}$$

$$b = \begin{bmatrix} 10 \\ 0 \\ 0 \\ -10 \end{bmatrix} \quad l = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad u = \begin{bmatrix} 7 \\ 8 \\ 6 \\ 10 \end{bmatrix} \quad c = \begin{bmatrix} 5 \\ 7 \\ 3 \\ 4 \end{bmatrix} \quad x = \begin{bmatrix} x_{12} \\ x_{13} \\ x_{24} \\ x_{34} \end{bmatrix}$$

8.3.3 Redes Residuais

No projeto e implementação de algoritmos é conveniente expressar o fluxo em termos de incrementos em relação a um fluxo factível, em vez de expressá-lo em termos absolutos. Aqui será introduzida a noção de *rede residual* ou *rede de fluxo remanescente* que descreve a capacidade de transmissão incremental. Existe uma relação de um-para-um entre a formulação do problema na rede original e na rede residual, de maneira que uma solução factível em uma formulação está diretamente relacionada com uma solução na outra formulação e ambas induzem o mesmo custo.

Dados uma rede $G = (V, A)$ e um fluxo $x^0 \in \mathbb{R}_+^{|A|}$, a *rede residual* é denotada por $G(x^0)$ e descreve a capacidade incremental de fluxo relativa ao fluxo x^0 já transportado através da rede. Considere o arco (i, j) e o fluxo inicial x_{ij}^0. Há duas possibilidades:

1) podemos enviar uma quantidade adicional $u_{ij} - x_{ij}^0$ de fluxo através do arco (i, j), incorrendo um custo unitário c_{ij}; e

2) podemos enviar de volta uma quantidade x_{ij}^0 através do arco (j, i), a custo $-c_{ij}$ por unidade.

Uma ilustração de rede residual $G(x^0)$ associada a um fluxo x^0 em uma rede $G = (V, A)$ é dada na Figura 8.12. Abaixo listamos os passos para cômputo da rede residual $G(x^0)$ tomando como base um fluxo inicial x^0:

1) substitua cada arco original (i, j) por dois arcos: (i, j) e (j, i);

2) o arco (i, j) tem custo c_{ij} e capacidade residual $r_{ij} = u_{ij} - x_{ij}^0$;

3) o arco (j, i) tem custo $-c_{ij}$ e capacidade residual $r_{ji} = x_{ij}^0$.

Proposição 8.1 *Um fluxo x é factível para a rede G se e somente se o fluxo correspondente $\bar{x}$, definido por $\bar{x}_{ij} - \bar{x}_{ji} = x_{ij} - x_{ij}^0$ e $\bar{x}_{ij}\bar{x}_{ji} = 0$, é factível na rede residual $G(x^0)$. Além disso, $c^T x = \bar{c}^T \bar{x} + c^T x^0$.*

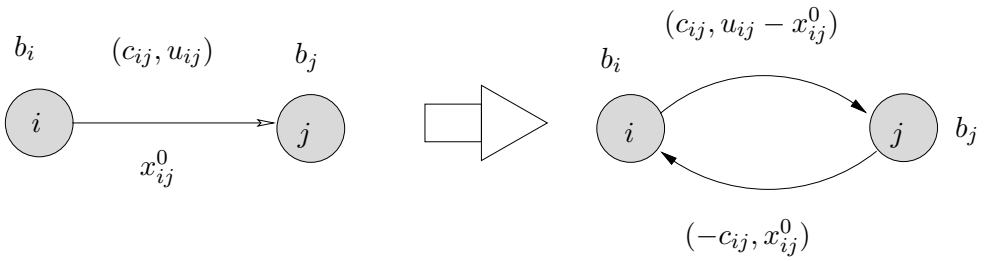


Figura 8.12: Exemplo de rede residual.

Prova: Seja x um fluxo factível para G. Seja $\bar{x}$ um fluxo residual para a rede $G(x^0)$ tal que $\bar{x} \geq 0$ e fazendo para todo $(i,j) \in A$:

$$\bar{x}_{ij} - \bar{x}_{ji} = x_{ij} - x^0_{ij} \tag{8.11}$$

$$\bar{x}_{ij}\bar{x}_{ji} = 0 \tag{8.12}$$

A condição 8.12 nos diz que apenas um dos arcos tem fluxo não nulo. Se $x_{ij} \geq x^0_{ij}$, então faça $\bar{x}_{ij} = x_{ij} - x^0_{ij}$ e $\bar{x}_{ji} = 0$; uma vez que $x_{ij} \leq u_{ij}$ temos que $\bar{x}_{ij} \leq u_{ij} - x^0_{ij} = r_{ij}$ e portanto $\bar{x}_{ij}$ satisfaz a restrição de capacidade do arco (i,j) na rede residual. Caso contrário, se $x_{ij} < x^0_{ij}$, então faça $\bar{x}_{ji} = x^0_{ij} - x_{ij}$ e $\bar{x}_{ij} = 0$; uma vez que $0 \leq x_{ij} \leq x^0_{ij}$, segue que $\bar{x}_{ji} = x^0_{ij} - x_{ij} \geq 0$ e $\bar{x}_{ji} < x^0_{ij} = r_{ji}$, portanto $\bar{x}_{ji}$ satisfaz a restrição de capacidade do arco (j,i) na rede residual. O desenvolvimento acima mostra que se o fluxo x é factível em G, então o fuxo residual $\bar{x}$ obtido conforme (8.11) e (8.12) é factível na rede residual $G(x^0)$. De forma similar, pode ser mostrado que para um fluxo residual $\bar{x}$ factível em $G(x^0)$ o fluxo x obtido de maneira a satisfazer as equações (8.11) e (8.12) é factível para a rede original G.

Resta estabelecer a relação entre o custo do fluxo x em G e o custo do fluxo $\bar{x}$ em $G(x^0)$. Para um fluxo x_{ij} na rede G, o custo no par de arcos (i,j) e (j,i) na rede residual $G(x^0)$ é $\bar{c}_{ij}\bar{x}_{ij} + \bar{c}_{ji}\bar{x}_{ji} = c_{ij}\bar{x}_{ij} - c_{ij}\bar{x}_{ji} = c_{ij}(\bar{x}_{ij} - \bar{x}_{ji}) = c_{ij}(x_{ij} - x^0_{ij})$ que segue de (8.11). Logo, estabelecemos que $\bar{c}\bar{x} = cx - cx^0$. ■

De acordo com a proposição, podemos trabalhar com a rede residual $G(x^0)$ para um fluxo x^0 em vez de trabalhar com a rede original G. Uma vez conhecido um fluxo ótimo na rede residual podemos convertê-lo em um fluxo ótimo na rede original por meio da proposição.

Na Figura 8.13 é ilustrada uma rede G com fluxo x e sua respectiva rede residual $G(x)$.

8.3.4 Algoritmo de Cancelamento de Circuitos Negativos

Nesta seção apresentamos uma condição necessária e suficiente para otimalidade de um fluxo e desenvolvemos um algoritmo que gera uma seqüência de soluções convergente para uma solução ótima que satisfaz as condições.

Teorema 8.4 *(Condição de Otimalidade) Um fluxo factível x^* induz uma solução ótima para o problema de fluxo de custo mínimo se, e somente se, a rede residual $G(x^*)$ não contém nenhum circuito de custo negativo.*

Proof: Seja x um fluxo factível e suponha que $G(x)$ contém um ciclo de custo negativo. Então podemos enviar uma quantidade não nula de fluxo ao longo do ciclo e reduzir o

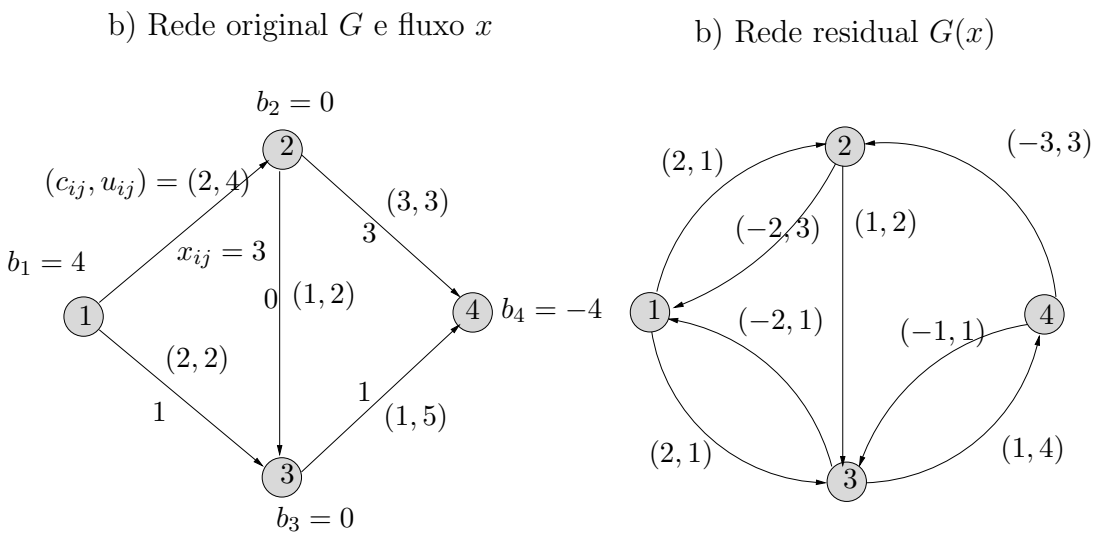


Figura 8.13: Exemplo de rede residual.

custo corrente $c^T x$ induzido pelo fluxo x. Logo, x não pode ser uma solução ótima e, portanto, se x^* é um fluxo ótimo então obrigatoriamente $G(x^*)$ não contém ciclo com custo negativo.

Seja x^* um fluxo factível tal que $G(x^*)$ não contém ciclo negativo. Suponha que existe um fluxo ótimo x^0 tal que $x^0 \neq x^*$. Então podemos decompor o vetor diferença $x^0 - x^*$ em no máximo $|A|$ ciclos de incremento de fluxo com respeito ao fluxo x^* tal que a soma dos custos ao longo destes ciclos equivale a $c^T(x^0 - x^*)$. Uma vez que a soma dos custos ao longo de cada um desses ciclos em $G(x^0)$ é não negativa, deduzimos que $c^T x^0 - c^T x^* \geq 0 \Rightarrow c^T x^0 \geq c^T x^*$. Mas como x^0 é um fluxo ótimo, $c^T x^0 \leq c^T x^*$. Concluímos que $c^T x^0 = c^T x^*$ e x^* também é um fluxo ótimo. ■

Um algoritmo pode ser obtido imediatamente a partir da condição de otimalidade estabelecida no Teorema 8.4. Basta encontrar um fluxo factível x^0, obter a rede residual $G(x^0)$, encontrar um ciclo w^0 de custo negativo em $G(x^0)$ e enviar a quantidade máxima de fluxo ao longo de w^0 obtendo x^1 e $G(x^1)$. Se $G(x^1)$ contém um ciclo de custo negativo, então repita o passo anterior, caso contrário pare, pois o fluxo corrente é uma solução ótima para o problema em questão. Os passos do algoritmo são dados abaixo.

Algoritmo: Cancelamento de Circuitos Negativos

Obtenha um fluxo viável x
Enquanto $G(x)$ contém um circuito negativo faça
    Utilize um algoritmo para encontrar um circuito negativo w
    $\delta = \min\{r_{ij} : (i, j) \in w\}$
    Aumente o fluxo a longo do circuito w de δ unidades
        e atualize $G(x)$
Fim-enquanto

Para aplicar o algoritmo são necessários dois algoritmos de suporte: um algoritmo para encontrar fluxo factível inicial e um algoritmo para encontrar ciclo negativo.

Como que se pode encontrar um fluxo factível? Fazendo uso de um algoritmo de fluxo máximo (enunciado na Seção 8.3) podemos encontrar um fluxo factível. Deixamos esses passos à cargo do leitor.

Como que se pode encontrar, caso exista, um ciclo de custo negativo na rede residual $G(x^k)$ correspondente a um fluxo x^k? Podemos fazer uso do algoritmo de Bellman-Ford [12] para caminhos mínimos em grafos direcionados com arcos de custo negativo, ou utilizar uma versão modificada do algoritmo de programação dinâmica de Floyd-Warshall [12] que encontra caminhos mínimos para todos os pares de vértices.

Complexidade do Algoritmo

Um limite superior no número de operações elementares executadas pelo algoritmo pode ser facilmente estabelecido. Seja $C = \max\{c_{ij} : (i,j) \in A\}$ assumindo que $c_{ij} \geq 0$[6] e seja $U = \max\{u_{ij} : (i,j) \in A\}$. Então mCU é um limite superior para o custo do fluxo inicial, onde $|A| = m$. Além disso, zero é um limite inferior para o custo do fluxo ótimo. Portanto, o algoritmo termina em $O(mCU)$ iterações onde $|V| = n$. Se utilizarmos um algoritmo com tempo de execução $O(nm)$ para detectar circuitos negativos[7], verificamos que o tempo de execução do algoritmo de cancelamento de circuitos negativos é $O(nm^2CU)$. Podemos dizer que o algoritmo é de tempo polinomial?

Exemplo de Execução do Algoritmo

Aqui aplicamos o algoritmo de cancelamento de circuito negativo ao problema dado na Figura 8.13. Tomando como fluxo inicial $x^0 = (x^0_{12}, x^0_{13}, x^0_{23}, x^0_{24}, x^0_{34}) = (3, 1, 0, 3, 1)$, obtemos a rede residual $G(x^0)$ dada na Figura 8.14. O ciclo de custo negativo w^0 está indicado na figura com linhas tracejadas. Enviando o fluxo de $\delta(w^0)$ unidades ao longo de w^0, obtemos a rede residual $G(x^1)$ dada na Figura 8.15. Para esta rede residual identificamos o circuito w^1, indicado por linhas tracejadas, com custo $c(w^1) = -1$ e capacidade $\delta(w^1) = 1$. Enviando a capacidade máxima ao longo de w^1, obtemos a rede residual $G(x^2)$ ilustrada na Figura 8.16. A rede residual $G(x^2)$ contém um ciclo de custo negativo w^2. Enviando $\delta(w^2) = 1$ unidades ao longo de w^2 obtemos a rede residual $G(x^3)$ indicada na Figura 8.17. Visto que $G(x^3)$ não possui ciclo negativo, concluímos que ela induz uma solução ótima. Para obter o fluxo ótimo x^3 a partir de $G(x^3)$ podemos fazer uso da Proposição 8.1. Obetmos então $x^3_{1,2} = 2$, $x^3_{1,3} = 2$, $x^3_{2,3} = 2$, $x^3_{2,4} = 0$ e $x^3_{3,4} = 4$.

8.4 Matrizes Totalmente Unimodulares

Considere o problema inteiro abaixo:

$$\begin{array}{lll} IP: & \text{Max} & c^T x \\ & \text{S.a}: & \\ & & Ax \leq b \\ & & x \in \mathbb{Z}^n_+ \end{array}$$

[6] Observe que sempre é possível eliminar arcos com custo negativo; basta pré-enviar a quantidade u_{ij} de fluxo através do arco (i,j) se $c_{ij} < 0$.

[7] O algoritmo de Bellman-Ford encontra um ciclo de custo negativo em tempo $O(nm)$.

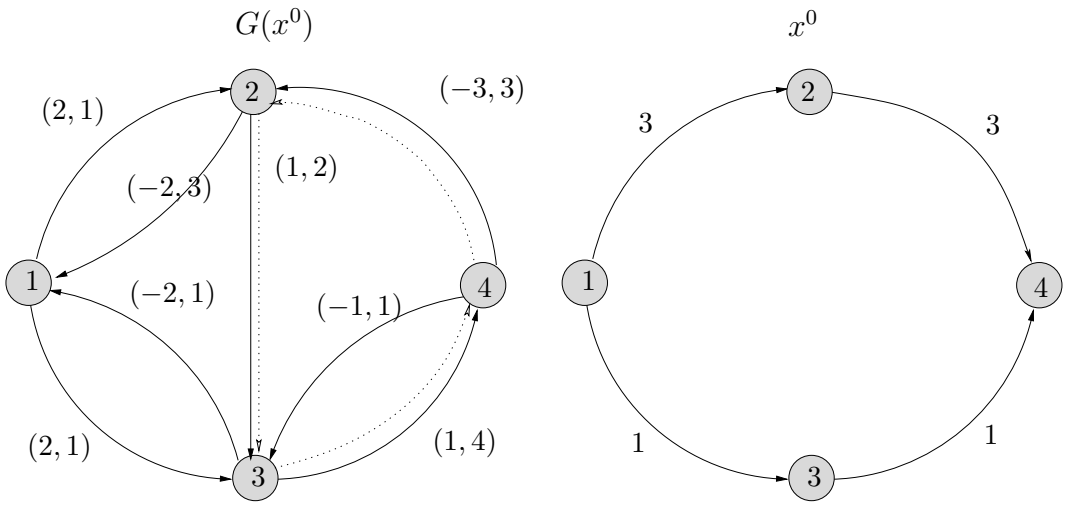


Figura 8.14: Primeira iteração do algoritmo de cancelamento de circuito negativo. A figura mostra a rede residual $G(x^0)$, o circuito com custo negativo em linhas tracejadas e o fluxo x^0. O circuito é dado por $w^0 = ((4,2),(2,3),(3,4))$ que tem custo $c(w^0) = -1$ e capacidade máxima $\delta(w^0) = 2$.

cujos dados (A, b) são inteiros com $A \in \mathbb{Z}^{m\times n}$. Sob quais condições podemos esperar que a relaxação linear LP de IP, $\{max\ c^T x : Ax \leq b,\ x \in \mathbb{R}^n_+\}$, terá solução inteira? A partir da teoria da programação linear, sabemos que as soluções básicas podem ser expressa na forma:

$$x = (x_B, x_N) = (B^{-1}b, 0)$$

onde B é uma submatriz $m \times m$ não-singular da matriz (A, I) e I é uma matriz identidade de dimensão $m \times m$.

Proposição 8.2 *(Condição Suficiente) Se a base ótima B tem determinante $det(B) = \pm 1$, então a relaxação linear resolve IP.*

Prova: Pela regra de Cramer, $B^{-1} = \frac{adj(B)}{det(B)}$ onde $adj(B)$ é a matriz adjunta de B[8]. As entradas de $adj(B)$ são produtos dos termos de B, portanto, $adj(B)$ é uma matriz de inteiros e $det(B) = \pm 1$, o que nos leva a concluir que $B^{-1}b$ é inteiro para qualquer b inteiro. □

Definição 8.1 *Uma matriz A é totalmente unimodular (TU) se toda a submatriz quadrada de A tem determinante -1, $+1$ ou 0.*

Dois exemplos de matrizes que não são TU:

$$A_1 = \begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix} \quad A_2 = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix}$$

[8]A matriz adjunta $adj(B)$ é a transposta da matriz cofatora $cof(B)$ de B, i.e, $adj(B) = cof(B)^T$. A entrada $cof(B)_{ij}$ é o determinante da matriz obtida a partir de B removendo a linha i e a coluna j e multiplicando o resultado por $(-1)^{i+j}$.

pois $det(A_1) = 2$ e $det(A_2) = 2$. Dois exemplos de matrizes TU são:

$$A_1 = \begin{pmatrix} 1 & -1 & -1 & 0 \\ -1 & 0 & 0 & 1 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \quad A_2 = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Note que se A é TU então $a_{ij} \in \{+1, -1, 0\}$ para todo i, j.

Proposição 8.3 *Uma matriz A é TU se e somente se:*

- *a matriz transposta A^T é TU; e*
- *a matriz (A, I) é TU.*

A condição suficiente dada pela proposição a seguir é simples e nos permite verificar que uma das matrizes acima é TU.

Proposição 8.4 *(Condição suficiente) A matriz A é TU se:*

i. $a_{ij} \in \{+1, -1, 0\}$ *para todo i, j;*

ii. cada coluna contém no máximo dois coeficientes não nulos, $\sum_{i=1}^{m} |a_{ij}| \leq 2$*; e*

iii. existe uma partição (M_1, M_2) do conjunto M das linhas tal que cada coluna j que contenha dois coeficientes não nulos satisfaz $\sum_{i \in M_1} a_{ij} - \sum_{i \in M_2} a_{ij} = 0$.

Prova: Suponha que A não é TU e seja B a menor submatriz quadrada de B para a qual $det(B) \notin \{0, 1, -1\}$. B não pode conter uma coluna com uma única entrada não nula, pois B não seria mínima. Portanto, B contém duas entradas não nulas em cada coluna. Pela condição (iii), adicionamos as linhas de M_1 e subtraímos as linhas de M_2 de forma que obtemos o vetor nulo e, portanto, $det(B) = 0$, o que constitui uma contradição. □

Retornando à questão de IP, fica claro que quando A é TU a relaxação linear LP de IP produz a solução ótima.

Proposição 8.5 *O programa linear* $\max\{c^T x : Ax \leq b, x \in \mathbb{R}^n_+\}$ *tem solução ótima inteira para um vetor b inteiro, para o qual existe uma solução ótima de valor finito, se e somente se A é TU.*

A Figura 8.18 ilustra uma rede cuja matriz de incidência é dada abaixo. Observe que a matriz de incedência N de um grafo direcionado $G = (V, A)$ satisfaz as condições suficientes estabelecidas pela Proposição 8.4. Portanto, uma matriz de incidência é totalmente unimodular[9].

[9]Basta fazer $M_1 = M$ e $M_2 = \emptyset$.

	x_{12}	x_{14}	x_{23}	x_{31}	x_{32}	x_{35}	x_{36}	x_{45}	x_{51}	x_{53}	x_{65}	$b(i)$	
1	1	1	0	-1	0	0	0	0	-1	0	0	=	3
2	-1	0	1	0	-1	0	0	0	0	0	0	=	0
3	0	0	-1	1	1	1	1	0	0	-1	0	=	0
4	0	-1	0	0	0	0	0	1	0	0	0	=	-2
5	0	0	0	0	0	-1	0	-1	1	1	-1	=	4
6	0	0	0	0	0	0	-1	0	0	0	1	=	-5

8.5 Referências

São inúmeras as aplicações de modelos e algoritmos de fluxo em redes, em particular os modelo de fluxo de custo mínimo. Neste capítulo tivemos a oportunidade de apreciar alguns problemas de cunho teórico-prático. Tais algoritmos são rotineiramente aplicados na resolução de subproblemas resolvidos como etapas de outros problemas mais complexos. Nos exercícios, por exemplo, podemos descobrir uma restrição de eliminação de sub-rota não satisfeita por uma solução candidata ao problema do caixeiro viajante a partir da solução de um problema de fluxo em redes.

Uma das referências mais atuais e completas em fluxo em redes é o texto de Ahuja, Magnanti e Orlin [3]. Nele são apresentadas várias aplicações e algoritmos para problemas que tem início com o problema de fluxo máximo, passam pelo problema de fluxo de custo mínimo, e também tratam de generalizações envolvendo funções objetivo convexas, fluxos distintos (*multicommodity network flow problems*) e também relaxação Lagrangeana. Os algoritmos são tratados de forma gradual, começando com algoritmos inspirados em princípios intuitivos, cobrindo algoritmos práticos e algoritmos com tempo de execução polinomial.

Outro texto recomendado é o livro de Lawler [34] que, embora mais antigo, tem uma abordagem mais sucinta e direta. Este também faz um aprofundamento na teoria de Matroids que está relacionada a matrizes totalmente unimodulares e algoritmos gulosos

8.6 Exercícios

EX 8.1: Obtenha o problema dual do problema de fluxo em redes.

EX 8.2: Formule o problema de caminhos mínimos de um vértice s para um vértice t como um problema de fluxo em redes. Formule o problema de encontrar a árvore de caminhos mínimos a partir de um vértice s para os demais vértices do grafo.

EX 8.3: Aplique o algoritmo de cancelamento de circuito negativo ao problema de fluxo em rede de custo mínimo especificado na Figura 8.19. (Encontre um fluxo viável x_0, obtenha a rede residual $G(x_0)$ e depois obtenha fluxos $x_1, x_2, \ldots, x^*$ até que a rede residual $G(x^*)$ não contenha nenhum circuito com custo negativo.)

EX 8.4: Demonstre a verdade ou falsidade da seguinte afirmação: suponha que todas as demandas e suprimentos (i.e., os valores de b_i) bem como as capacidades dos

arcos são valores inteiros e pares, em um problema de fluxo em rede de custo mínimo. Então, existe um fluxo ótimo x^* onde cada fluxo $x^*_{i,j}$ é um número par.

EX 8.5: O conjunto de soluções ótimas para um problema qualquer de fluxo em redes, de custo mínimo, se altera se multiplicarmos o custo de cada arco por uma constante k? O conjunto se altera se adicionarmos uma constante k?

EX 8.6: Em um problema de fluxo em redes, suponha que além das capacidades dos arcos os nós também apresentam limites superiores para o fluxo de entrada. Seja w_i o fluxo máximo de entrada para o nó i, $i \in V$. Como que você resolveria esta generalização do problema de fluxo em redes?

EX 8.7: O problema de fluxo máximo em redes tem como dados uma rede $G = (V, A)$, a capacidade u_{ij} de transmissão para cada arco $(i, j) \in A$, um nó origem s e um nó destino t. O problema se resume a encontrar o fluxo x_{ij} através de cada arco (i, j) que maximize a quantidade de fluxo enviada de s para t, enquanto satisfazendo às restrições de capacidade nos arcos. Em programação matemática, o problema fica:

$$\begin{array}{ll} \text{Maximize} & \displaystyle\sum_{\{j:(s,j)\in A\}} x_{sj} \\ \text{Sujeito a :} & \\ & \displaystyle\sum_{\{j:(i,j)\in A\}} x_{ij} - \sum_{\{j:(j,i)\in A\}} x_{ji} = 0 \qquad \forall i \in V - \{s,t\} \\ & 0 \le x_{ij} \le u_{ij} \qquad\qquad\qquad\qquad\qquad\quad \forall (i,j) \in A \end{array}$$

Mostre como se pode encontrar um fluxo factível para o problema de fluxo em redes de custo mínimo fazendo uso do problema de fluxo máximo.

EX 8.8: No problema custo-mínimo fluxo-máximo definido em uma rede direcionada $G = (V, A)$, desejamos enviar o maior fluxo possível do nó origem s para o nó destino t, mas com o menor custo possível. Em outras palavras, dentre todos os fluxos máximos, encontre aquele de menor custo. Tarefas:

1) mostre como formular qualquer problema de fluxo em redes de custo mínimo como um problema custo-mínimo fluxo-máximo;

2) mostre como o problema de custo-mínimo fluxo-máximo pode ser resolvido por meio de uma redução ao problema de custo mínimo.

EX 8.9: No problema de caminhos mínimos com pedágio é dado um grafo direcionado $G = (V, A)$, o peso dos arcos c_{ij} correspondendo ao custo do combustível necessário para realizar o percurso da cidade i até j, e o valor do pedágio p_j cobrado a cada cidade j. Desejamos encontrar um caminho de menor custo (combustível e pedágio) da cidade s até a cidade t. Pedágio p_j é coletado na entrada e na saída da cidade j. Formule este problema como um problema de fluxo em redes de custo mínimo.

EX 8.10: Para um grafo direcionado $G = (V, E)$, o problema do caixeiro viajante

pode ser colocado em programação matemática como:

$$
\begin{aligned}
P_T: \quad \text{Minimize} \quad & \sum_{(i,j)\in E} c_{ij}x_{ij} \\
\text{Sujeito a :} \quad & \\
& \sum_{\{j:(i,j)\in E\}} x_{ij} = 1,\ \forall i \in V \\
& \sum_{\{j:(j,i)\in E\}} x_{ji} = 1,\ \forall i \in V \\
& \sum_{(i,j)\in(S,V-S)} x_{ij} \geq 1,\ \forall S \subset V, 2 \leq |S| \leq |V|-2 \\
& x_{ij} \in \{0,1\},\ \forall (i,j) \in E
\end{aligned}
\tag{8.13}
$$

$(S, V-S) = \{(i,j) \in E : i \in S, j \in V-S\}$ é o corte de G induzido por S. Note que a formulação P_T tem um número exponencial de restrições de eliminação de subrotas, assim não é prático explicitar as desigualdades de (8.13). Em vez disso, métodos mais práticos ignoram a terceira família de restrições de (8.13), transformam a quarta família de retrições em $0 \leq x_{ij} \leq 1$ e adicionam restrições de eliminação de sub-rotas sob demanda. Denote este problema simplificado por P'_T.

Seja $x \in \mathbb{R}^{|E|}$ uma solução factível para P'_T. Desenvolva um algoritmo eficiente para encontrar um corte $(S, V-S)$ tal que a solução aproximada x não satisfaça uma restrição de eliminação de sub-rota. O algoritmo deve também indicar quando um corte não existe.

EX 8.11: Obtenha o dual do problema de fluxo máximo.

EX 8.12: Considere um problema cujas restrições são dadas por:

$$
\begin{array}{lcll}
x_1 & \leq & 1 & \\
x_n & \geq & 0 & \\
x_j & \geq & z_j, & j = 1, \ldots, n-1 \\
x_{j+1} & \leq & z_j, & j = 1, \ldots, n-1
\end{array}
\tag{8.14}
$$

Podemos afirmar que o sistema de equações (8.14) forma uma matriz totalmente unimodular? Justique sua resposta.

EX 8.13: Mostre que o sistema de equações (8.15) induz uma matriz de restrições totalmente unimodular.

$$
\begin{array}{lcl}
\lambda_i^L + \lambda_i^R & = & y_i,\ i = 1, \ldots, n \\
\sum_{i=1}^{n} y_i & = & 1 \\
\lambda_i^L, \lambda_i^R, y_i & \geq & 0,\ i = 1, \ldots, n
\end{array}
\tag{8.15}
$$

Este sistema de equações surge da linearização por partes de uma função não linear $f(x)$, onde $x = \sum_{i=1}^{n}[a_{i-1}\lambda_i^L + a_i\lambda_i^R]$, $f(x) \approx \sum_{i=1}^{n}[f(a_{i-1})\lambda_i^L + f(a_i)\lambda_i^R]$, e $a_0 < a_1 < \ldots < a_n$ [41].

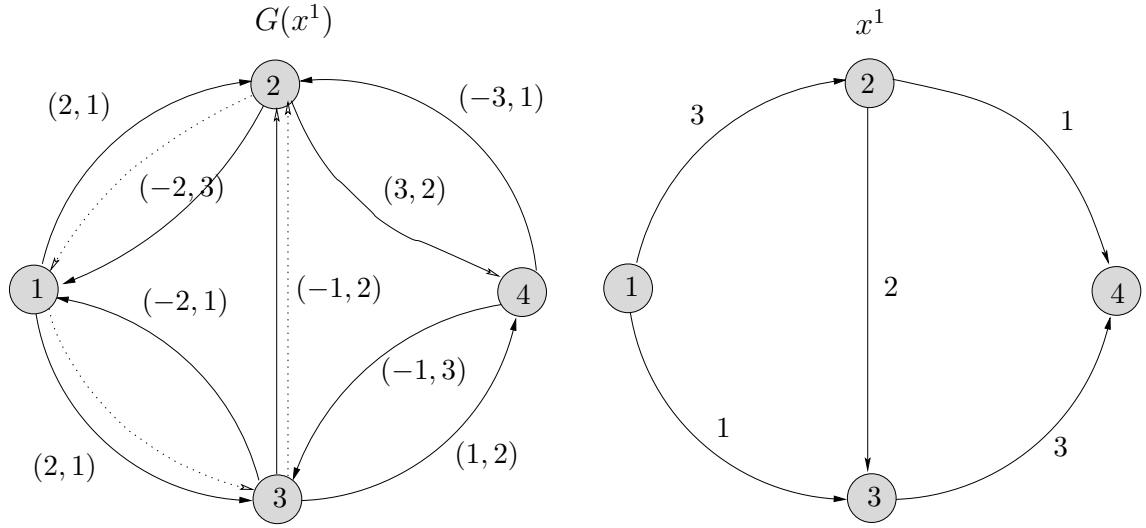


Figura 8.15: Segunda iteração do algoritmo de cancelamento de circuito negativo. A figura mostra a rede residual $G(x^1)$, o ciclo $w^1 = ((3,2),(2,1),(1,3))$ com custo $c(w^1) = -1$ e capacidade máxima $\delta(w^1) = 1$, e o fluxo x^1.

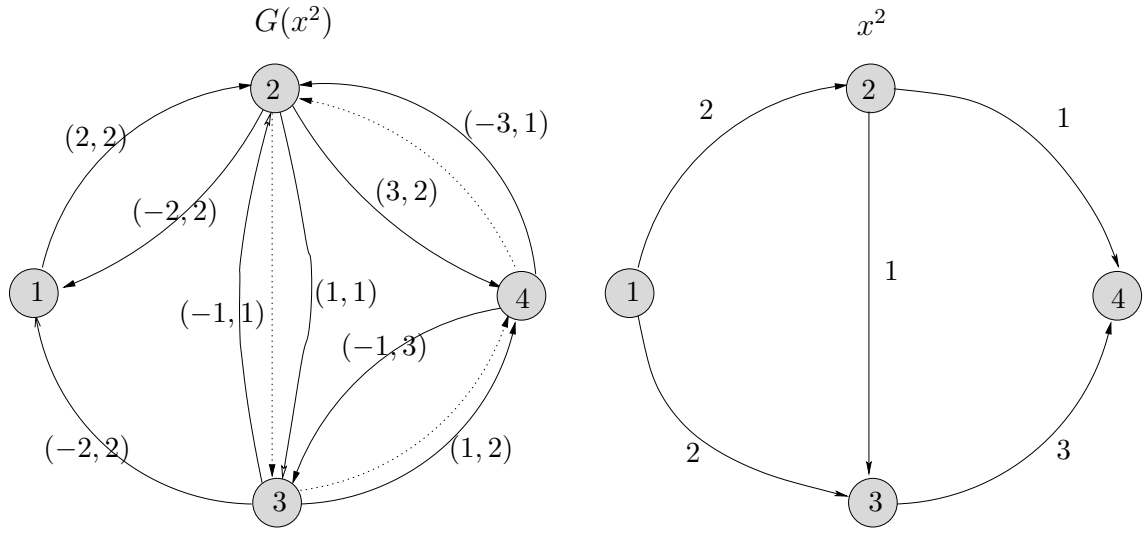


Figura 8.16: Terceira iteração do algoritmo de cancelamento de circuito negativo. Na figura é dada a rede residual $G(x^2)$ e o respectivo fluxo x^2. Também é indicado o circuito $w^2 = ((2,3),(3,4),(4,2))$ com custo $c(w^2) = -3$ e capacidade $\delta(w^2) = 1$.

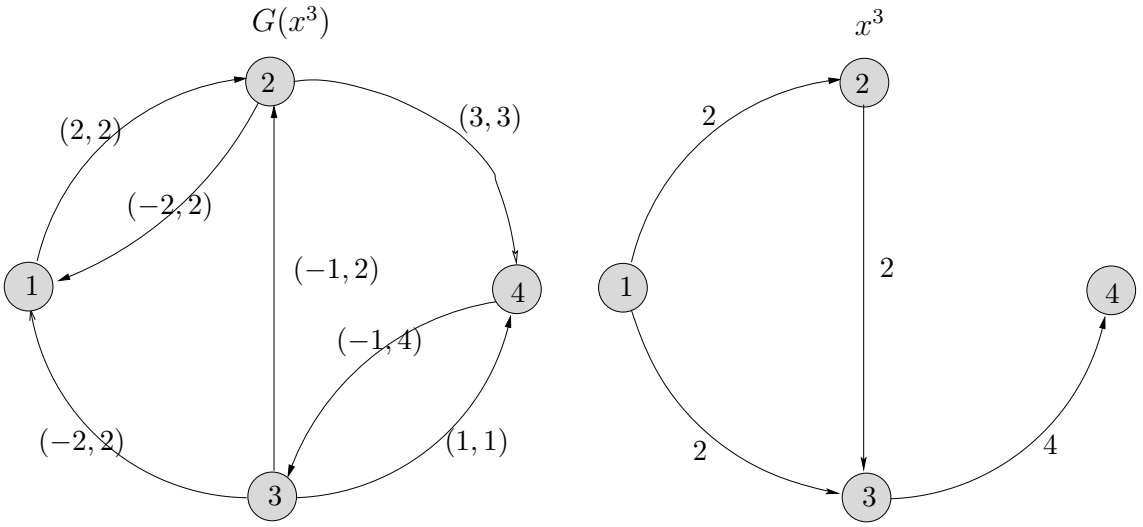


Figura 8.17: Fluxo ótimo x^3 e rede residual $G(x^3)$. Não há ciclo de custo negativo.

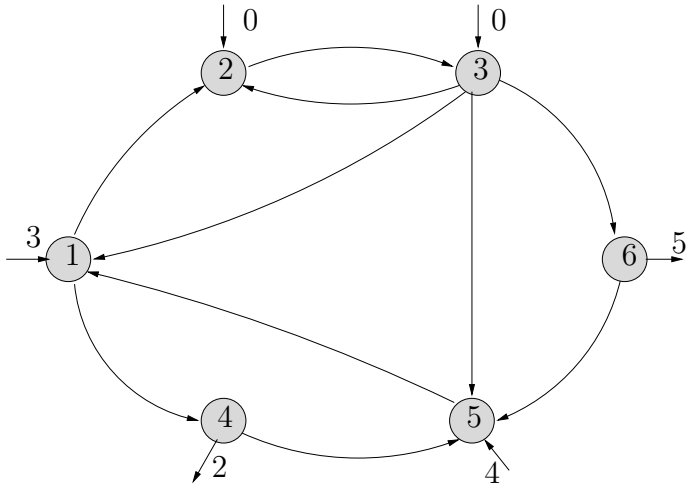


Figura 8.18: Propriedade de unimodularidade das matrizes de incidência de grafos. Um arco sem nó de origem indica fluxo injetado no nó destino, enquanto que um arco sem nó de destino indica fluxo consumido pelo nó origem.

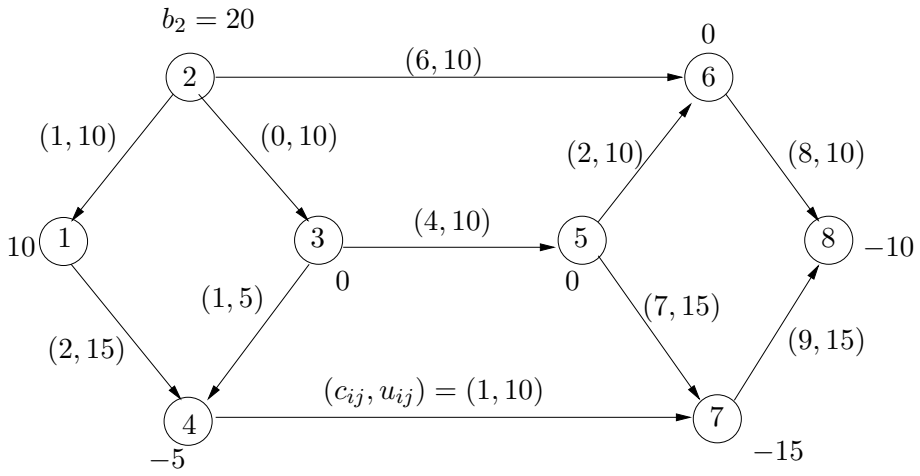


Figura 8.19: Grafo $G = (V, A)$ com a especificação do problema de fluxo em rede.

Capítulo 9

Linguagens de Modelagem

Neste capítulo apresentamos os elementos fundamentais de duas linguagens de modelagem que permitem especificar problemas de otimização em uma linguagem muito semelhante à programação matemática. Em particular, desenvolveremos os conceitos básicos das linguagens AMPL e Mosel. Tais linguagens procuram separar o modelo dos dados, dessa forma permitindo que um modelo possa ser utilizados na resolução de diferentes instâncias de uma mesma classe de problemas. Elas também são responsáveis pela parte de pré-processamento e interface com algoritmos de otimização, tais como ILOG CPLEX, MINOS e XPress-MP.

9.1 Linguagem Mosel

A linguagem Mosel faz parte do pacote de software de otimização Xpress-MP. Xpress-MP é uma ferramenta de software para modelagem matemática e solução de problemas de otimização linear (PL), quadrática (PQ) e linear inteira (PI). As ferramentas do Xpress-MP compreendem uma coleção de interfaces, objetivando atender necessidades de usuários diversos e permitindo a solução de problemas bem como a integração com outros produtos de software. Os dois componentes básicos do Xpress-MP são Xpress-Mosel e Xpress-Optimizer, mas há outros componentes também relevantes destacando-se Xpress-IVE, Console Xpress e Xpress-MP Libraries.

1) Xpress-Mosel: É um ambiente para modelagem e solução de problemas, tomando como entrada um modelo que descreve um problema em programação matemática (PL, PQ, PI), escrito na linguagem de modelagem Mosel. Mosel permite separar modelo dos parâmetros que definem uma instância e tem facilidades de transferência de dados. Por exemplo, dados podem ser transferidos através de arquivos tipo texto e em formato ODBC.

2) Xpress-Optimizer: Optimizer é a parte central da ferramenta Xpress-MP, representando décadas de pesquisa e desenvolvimento de métodos de solução de problemas lineares, quadráticos e inteiros mistos. A ferramenta dispõe de algoritmos de pré-processamento para simplificação de modelos, métodos de geração de cortes, etc.

3) Xpress-IVE: Xpress-IVE (*The Xpress Interactive Visual Environment*) é um ambiente para modelagem e solução integrada de problemas de otimização em sistemas MS-Windows. IVE apresenta Mosel dentro de uma interface gráfica amigável com editor embutido.

4) Console Xpress: Consiste em interfaces tipo texto para Mosel e Optimizer. Este módulo permite a carga de modelos armazenados em arquivos tipo texto e resolução de problemas, podendo ser executado em plataformas computacionais diversas, incluindo Unix e MS-Windows. O Console Xpress pode ser aplicado em ambientes de produção fazendo uso de arquivos *shell-script*.

5) Xpress-MP Libraries: Para implementações especializadas, bibliotecas Xpress-MP são oferecidas para prover acesso direto a Mosel e Optimizer a partir de código C/C++, Java e Visual Basic implementado e customizado pelo usuário. A vantagem principal das bibliotecas é a possibilidade de interagir diretamente com a funcionalidade de Xpress-MP e, dessa forma, desenvolver aplicações customizadas como, por exemplo, heurísticas projetadas para problemas particulares e algoritmos tipo *branch-and-cut*.

Uma versão limitada no número de variáveis e restrições, mas completa em termos de funcionalidade, do Xpress-IVE pode ser obtida para uso acadêmico e educacional no site da Dash Optimization: `http://www.dashoptimization.com`.

9.1.1 Qual Interface Devemos Utilizar?

Esta decisão depende das necessidades do usuário.

Usuários iniciantes devem preferencialmente utilizar Xpress-IVE, uma vez que esta é a interface mais simples para o software Xpress-MP. O ambiente integrado permite rápida modelagem ao mesmo tempo que facilita a comunicação com os demais componentes do Xpress-MP

Já em Ambientes de produção, a interface Console Xpress é mais adequada pois esta permite que se desenvolva aplicações tipo *batch*. Console Xpress provê uma interface tipo texto poderosa para os componentes Xpress-MP, permitindo desenvolvimento interativo e processamento *batch* de modelos de usuários de plataformas diversas.

Por outro lado, as bibliotecas devem ser reservadas para os usuários avançados que desejem desenvolver e testar algoritmos (e.g., heurísticas inteligentes e algoritmos *branch-and-bound*).

9.1.2 Resolvendo um Problema

Aqui descrevemos os passos necessários para se resolver um problema através da interface Mosel. Considere o problema de programação linear abaixo:

$$\begin{array}{ll} \text{Maximize} & a + 2b \\ \text{Sujeito a :} & 3a + 2b \leq 400 \\ & a + 3b \leq 200 \\ & a, b \geq 0 \end{array}$$

Abaixo segue a especificação do problema acima na linguagem Mosel.

```
model Simple
  uses ‘‘mmxprs’’
  declarations
    a:  mpvar
    b:  mpvar
  end-declarations

  primeira := 3*a + 2*b <= 400
  segunda := a + 3*b <= 200
  lucro := a + 2*b

  maximize(lucro)

  writeln(‘‘O lucro é:  ’’, getobjval)

end-model
```

O modelo acima é quase auto-explicativo. A diretiva `uses` é utilizada para declarar a biblioteca de otimização a ser utilizada. Na seção `declarations` são declaradas as variáveis do modelo. As restrições, denominadas `primeira` e `segunda`, são descritas na seqüência sendo ambas as desigualdades do tipo "$\leq$". A função objetivo é denominada `lucro` e utilizada pelo comando `maximize`, que executará o algoritmo de otimização adequado para o problema, buscando uma solução que respeite as restrições e maximize o lucro. No momento em que a solução ótima for encontrada, o valor da função objetivo ótima é impresso na saída padrão através da diretiva `writeln`.

A solução de um problema em Mosel é um processo de 3 estágios consistindo em i) compilar o programa modelo, ii) carregar o programa compilado, e iii) resolver o programa. Os comandos que realizam estas três etapas são: `compile`, `load`, e `run` respectivamente.

Abaixo ilustramos a execução dos passos para se resolver o problema de interesse. O modelo está armazenado no arquivo "`simple.mos`".

```
C:> mosel
** Xpress-Mosel **
© Copyright Dash Associates 1998-2002
> compile simple
Compiling simple
> load simple
> run
O lucro é 171.429
Returned value:  0
```

Para obter informações sobre os comandos disponíveis e parâmetros, simplesmente digite o comando `help` na janela de entrada da interface Mosel.

9.1.3 Indo Mais Longe

O formato MPS é um padrão para representação e armazenamento de problemas de programação matemática. Vários otimizadores aceitam problemas especificados neste

formato. Detalhes sobre o formato MPS podem ser obtidos nos apêndices do manual de usuário do GNU solver. Mosel e Xpress-optimizer não podiam ser diferentes e ambos suportam a entrada e saída de problemas no formato MPS. Para gerar o arquivo MPS correspondendo ao problema escrito em Mosel, basta executar o comando:

```
> export -m simple.mps
```

Mosel também é capaz de gerar o problema no formato LP, que pode ser resolvido com certos pacotes de otimização como, por exemplo, `lp_solve` (`ftp://ftp.es.ele.tue.nl/pub/lp_solve`) e `glpk` da GNU (`http://www.gnu.org/software/glpk/glpk.html`). Para gerar o arquivo LP correspondente ao problema escrito em Mosel, execute o comando:

```
> export simple.lp
```

9.1.4 Trabalhando com o Optimizer

Nos casos anteriores, carregamos modelos dentro do ambiente Mosel e invocamos optimizer como um módulo da biblioteca. Optimizer pode também ser executado isoladamente. Ele aceita problemas armazenados em arquivos tipo matriz, em formato MPS ou LP. Com um arquivo "pb.mat" válido, tudo que precisa ser feito é carregar a matriz completamente no optimizer e maximizar/minimizar a função objetivo. Abaixo segue a seqüência de comandos que carregam e resolvem um problema armazenado em um arquivo tipo MPS:

```
C:> optimizer
Xpress-MP Integer Barrier Optimizer Release xx.yy
© Copyright Dash Associates 1998-2002
Enter problem name> simple
> readprob
      ⋮
> maxim
      ⋮
> writeprtsol (ou prtsol)
> quit
```

9.1.5 Construindo um Primeiro Modelo

Trataremos aqui de flexibilidades da modelagem com Mosel bem como a separação da estrutura do modelo e dos dados, que juntos formam uma instância do problema. Utilizaremos o problema da mochila para ilustrar tal flexibilidade.

$$\begin{aligned} &\text{Maximize} && \sum_{i=1}^{n} c_i x_i \\ &\text{Sujeito a :} && \\ & && \sum_{i=1}^{n} a_i x_i \leq b \\ & && x_i \in \{0,1\},\ i = 1, \ldots, n \end{aligned}$$

Tabela 9.1: Dados do problema da mochila

Objeto	Peso	Valor
Camera	2	15
Colar	20	100
Vaso	20	90
Quadro	30	60
TV	40	40
Video	30	15
Bau	60	15
Tijolo	10	1

Considere a instância do problema da mochila obtida com os dados da Tabela 9.1.

A especificação Mosel do problema da mochila para os dados acima pode ser realizada como segue:

```
model mochila
uses ‘‘mmxprs’’

declarations
   camera, colar, vaso,
   quadro, tv, video, bau,
   tijolo :  mpvar
end-declarations

camera is_binary
colar is_binary
vaso is_binary
     ⋮
tijolo is_binary

PesoTotal := 2*camera + 20*colar + 20*vaso + 20*quadro
   + 40*tv + 30*video + 60*bau + 10*tijolo <= 102
ValorTotal := 15*camera + 100*colar + 90*vaso + 60*quadro
   + 40*tv + 15*video + 10*bau + 1*tijolo

maximize(ValorTotal)
writeln(‘‘O valor da função objetivo é ’’,getobjval)

end-model
```

A construção de modelos como visto acima é conveniente para problemas com um número pequeno de variáveis, mas isso pode se tornar inviável em problemas de grande porte. Foi tedioso, por exemplo, ter que especificar que cada variável é binária. É usual na modelagem, mesmo de problemas pequenos, o uso de vetores de variáveis ou variáveis subscritas, como também são conhecidas. Tal política de modelagem facilita enormemente a especificação de modelos e futuras alterações. Abaixo segue a segunda versão Mosel do problema da mochila.

```
model mochila2
uses ‘‘mmxprs’’

declarations
  Itens = 1..8
  Peso:  array(Itens) of real
  Valor:  array(Itens) of real
  x :  array(Itens) of mpvar
end-declarations

!  Itens
Peso:= [2, 20, 20, 30, 40, 30,
      60, 10]
Valor:= [15, 100, 90, 60, 40,
      15, 10, 1]

!  Todas as variáveis x são inteiras
forall(i in Itens) x(i) is_binary
!  Objetivo
ValorTotal := sum(i in Itens) x(i)*Valor(i)
!  Restrição de capacidade da mochila
PesoTotal := sum(i in Itens) x(i)*Peso(i) <= 102

maximize(ValorTotal)

writeln(‘‘O valor da função objetivo é ’’, getobjval)
forall(i in Itens) writeln(‘‘x(’’,i,‘‘) =’’,getsol(x(i)))

end-model
```

A modelagem com vetores trouxe algumas facilidades, dentre elas destacam-se:

1) Itens: é um conjunto indexado para os elementos do vetor.

2) forall: permite varrer elementos de um vetor indexado.

3) sum: permite somar elementos de um vetor indexado.

9.1.6 Usando Cadeias de Caracteres como Índices

O modelo anterior é consideravelmente mais simples do que o primeiro modelo para o problema da mochila, mas sua interpretação é mais difícil—temos que traduzir os identificadores em seus respectivos nomes. Contudo, podemos melhorar a legibilidade do código Mosel por meio da seguinte substituição:

```
Itens = {‘‘camera’’, ‘‘colar’’, ‘‘vaso’’, ‘‘quadro’’,
   ‘‘tv’’, ‘‘video’’, ‘‘bau’’, ‘‘tijolo’’}
```

9.1.7 Modelagem Versátil

Os exemplos acima são apenas alguns de um número de problemas similares, conhecidos como problemas da mochila. Seria útil generalizar o modelo acima, permitindo o aumento do número de itens e a alteração dos pesos/valores. Com o problema já especificado em termos de vetores, as dificuldades estão no acoplamento do modelo com os dados. Aqui introduzimos algumas definições que facilitam o entendimento dessas questões:

1) Modelos: são geralmente especificados usando símbolos para representar as diversas variáveis de decisão, bem como o relacionamento entre as variáveis por meio de desigualdades e igualdades.

2) Modelos Genéricos: são obtidos através de uma descrição sistemática dessas restrições e dos objetivos.

3) Instância: é o resultado da combinação do modelo genérico com um conjunto de dados, a qual pode ser otimizada.

Modelo: "`mochila3.mos`"

```
model mochila3
uses ‘‘mmxprs’’

declarations
   Itens = set of strings
   MAXWT: real
   Peso:  array(Itens) of real
   Valor:  array(Itens) of real
   x :  array(Itens) of mpvar
end-declarations

!  Carregar os dados a partir de arquivos externos
initializations from ‘‘mochila3.txt’’
   Itens MAXWT Peso Valor
end-initializations

!  Criar variáveis
forall(i in Itens) create(x(i))
!  Todas as variáveis x são binárias
forall(i in Itens) x(i) is_binary
!  Objetivo
ValorTotal := sum(i in Itens) x(i)*Valor(i)
!  Restrição de capacidade de peso da mochila
PesoTotal := sum(i in Itens) x(i)*Peso(i) <= MAXWT
maximize(ValorTotal)
writeln(‘‘O valor da função objetivo é ’’, getobjval)
end-model
```

Arquivo de Dados: "mochila3.txt"

```
Itens:  {‘‘camera’’, ‘‘colar’’, ‘‘vaso’’, ‘‘quadro’’,
   ‘‘tv’’, ‘‘video’’, ‘‘bau’’, ‘‘tijolo’’}
Peso:  [2, 20, 20, 30, 40, 30, 60, 10]
Valor:  [15, 100, 90, 60, 40, 15, 10, 1]
MAXWT: 120
```

9.2 Linguagem AMPL

AMPL pode ser vista como uma linguagem utilizada para descrever de uma forma declarativa problemas de planejamento, escalonamento e distribuição da produção e muitos outros problemas conhecidos em geral como problemas de otimização em larga escala ou programação matemática. A notação algébrica de AMPL e o seu ambiente de comandos interativo foram projetados para auxiliar na formulação de modelos, comunicar com uma variedade de pacotes de otimização e examinar o resultado de soluções. A flexibilidade de AMPL a torna ideal para prototipação rápida e desenvolvimento de modelos, enquanto que sua velocidade e generalidade provêem os recursos necessários para resolução em regime de produção.

AMPL é um entre vários sistemas de modelagem para otimização que são projetados tomando como base linguagens de modelagem algébricas. Essas linguagens empregam uma notação matemática muito semelhante à programação matemática, tipicamente utilizada para descrever problemas de otimização como a minimização (ou maximização) de uma expressão algébrica envolvendo variáveis de decisão algébricas, estando estas subjeitas a restrições expressas como igualdades e desigualdades entre expressões algébricas e variáveis de decisão. Interpretadores e interfaces para essas linguagem provêem suporte para simplificação e análise de modelos. Algumas linguagens também oferecem extensões para descrisão de métodos algorítmicos para atacar problemas difíceis por meio da resolução de subproblemas relacionados. MOSEL e AMPL são dois exemplos de linguagens de modelagem algébricas distribuídas comercialmente.

AMPL foi projetada para combinar e estender as habilidades expressivas de linguagens de modelagem, mas sem perder a facilidade de ser utilizada em aplicações elementares. AMPL é notável pela simplicidade e naturalidade de sua sintaxe e pela generalidade dos seus conjuntos e expressões de indexação. AMPL provê forte suporte à validação, verificação e análise de soluções ótimas através de um conjunto de alternativas para apresentação de dados e resultados. O pré-processador de AMPL é capaz de executar automaticamente transformações que reduzem o tamanho do problema e substituem variáveis. AMPL se distingue ainda pela continuidade do seu desenvolvimento que visa atender às necessidades dos usuários. Adições recentes incluem construtores de laços e testes para escrita de *scripts* na linguagem de comandos AMPL, bem como facilidades para definição e manipulação de vários problemas inter-relacionados.

No que segue apresentamos um exemplo de AMPL para o problema da mochila.

9.2.1 Modelo AMPL do Problema da Mochila

A descrição de um problema é feita através da especificação de três arquivos: o arquivo com o modelo genérico, o arquivo com os dados e o arquivo de comandos.

1) Modelo Genérico: Especificação das variáveis, restrições e função objetivo.

```
#---------------------------------------
# Problema da Mochila

set Itens;

param c {j in Itens}; # c[j] é o valor do item j
param w {j in Itens}; # w[j] é o peso do item j

param b >= 0;

var x {j in Itens} integer >= 0, <= 1;

# Função objetivo
maximize valor:  sum{j in Itens} c[j]*x[j];

# restrição capacidade da mochila
subject to mochila:
   sum{j in Itens} w[j]*x[j] <= b;
```

2) Arquivo de Dados: São explicitados os parâmetros de uma instância particular.

```
set Itens := camera colar vaso quadro
             tv video bau tijolo;

param:                  w   c :=
             camera     2   15
             colar      20  100
             vaso       20  9
             quadro     30  60
             tv         40  40
             video      30  15
             bau        60  10
             tijolo     10  1;

param b := 120;
```

3) Arquivo de Comandos: Diretivas para AMPL.

```
solve;
display valor;
display w;
display c;
display x;
```

9.2.2 Comentários

A linguagem AMPL rapidamente se tornou um padrão entre pesquisadores e engenheiros, hoje disponível na maior parte dos pacotes de otimização. Há um site exclusivamente dedicado à AMPL, `http://www.ampl.com`, onde estão depositados exemplos além de inúmeras informações e ponteiros. A propósito, existe um livro sobre AMPL e suas aplicações bem como uma versão educacional da linguagem que pode ser empregada para resolver problemas com um número pequeno de variáveis e restrições.

9.3 Estudo de Caso: Alocação de Rotas em Redes de Computadores

Com a rápida evolução da Internet em termos de desempenho e segurança, surgiu a exigência de maior qualidade dos serviços oferecidos pelas operadoras de telecomunicações. Novas políticas operacionais foram implementadas com estratégias de diferenciação de serviços, adicionando flexibilidade e eficiência à infra-estrutura de comunicações. O gerenciamento da qualidade de serviços (QoS) desponta neste cenário como uma alternativa vital ao sobre-provisionamento de recursos. Neste contexto, a Engenharia de Tráfego (ET) representa uma forma de aproveitar melhor os recursos de rede existentes, com consequente redução de custos [5].

Um elemento chave de suporte à ET em redes IP é a tecnologia *Multiprotocol Label Switching* (MPLS). A tecnologia MPLS possibilitou a extensão das funcionalidades do protocolo IP, favorecendo a implementação da engenharia de tráfego em redes IP graças à possibilidade de execução do roteamento explícito sobre a arquitetura MPLS. O roteamento explícito é a capacidade de encaminhamento de pacotes em caminhos virtuais previamente definidos, os chamados *Label Switched Paths* (LSPs).

Busca-se portanto melhorar a eficiência da rede através do uso parcimonioso de seus recursos, num cenário de operação dinâmica, com um impacto mínimo na arquitetura da infra-estrutura existente. Para tanto, faz-se necessária a modelagem formal do problema de engenharia de tráfego em programação matemática e o projeto de algoritmos eficientes.

9.4 Formulação do Problema

Um modelo inicial foi proposto em [17] e depois estendido em [18] de forma a prover uma política de admissão de requisições de transmissão (LSPs). Neste modelo busca-se a maximização da vazão global dos fluxos de dados injetados em uma topologia de rede. Para tanto são configurados caminhos de menor distância métrica entre origem

e destino dos fluxos de dados, bem como das larguras de bandas de LSPs, nos quais os fluxos de dados serão encaminhados. Estas configurações devem respeitar as restrições de largura de banda dos enlaces e atraso fim-a-fim máximo de cada fluxo de dados. Do ponto de vista da teoria de otimização, este problema é enquadrado como um *problema de caminhos mínimos sujeito à restrições* [23], sendo do tipo NP-completo [21].

Adotou-se uma política de diferenciação de serviços que depende de um controle de admissão e da configuração de largura de banda dos LSPs, baseadas em parâmetros de prioridade para cada fluxo de dados. Nesta política, na eminência de congestionamento da rede, os fluxos de baixa prioridade podem ter suas vazões reduzidas até zero (fluxos rejeitados) para garantir os requisitos de QoS dos fluxos de alta prioridade.

A largura de banda a ser configurada nos LSPs é modelada em níveis, ou seja, cada LSP será configurado com um desses possíveis níveis de largura de banda, necessários à alocação dos fluxos de dados. Estes níveis podem variar de um valor igual a zero até um valor máximo desejado. A discretização das larguras de banda a serem configuradas nos LSPs permite a formulação do problema em programação linear inteira, o que significa um tratamento computacional menos complexo do que num problema não-linear [44].

A formulação do problema de ET proposto tem como objetivo definir as características topológicas e lógicas da rede, a função objetivo, as variáveis de decisão e as famílias de restrições.

As características topológicas são modeladas por um grafo direcionado $G = (V, E)$, onde $V = \{1, \ldots, n\}$ corresponde aos nós da rede e $E = \{1, \ldots, m\}$ aos enlaces de transmissão. A capacidade de transmissão de um enlace (i, j) é μ_{ij} $Kbps$, sendo o custo administrativo c_{ij} correspondente ao atraso de transmissão provocado pelo mesmo. A função objetivo visa maximizar a vazão global dos fluxos de dados injetados na rede.

As variáveis de decisão permitem a seleção do nível de largura de banda a ser configurado em cada LSP, além da definição do caminho entre a fonte e o destino do fluxo de dados correspondente. As variáveis adotadas são: y_k^l denominada seletor de nível e x_{ij}^{kl} que determina se o LSP k está ou não roteado no enlace (i, j) para o l-ésimo nível de transmissão. Qualquer caminho que atenda as restrições poderá ser selecionado pela solução do problema.

As características lógicas da rede são como segue:

- o número de LSPs a serem configurados é K;
- o número de níveis de largura de banda a serem configurados no k-ésimo LSP é l_k; o número de níveis e a largura de banda de cada fluxo são definidos de acordo com critérios administrativos—o procedimento de otimização escolhe somente um nível para configurar cada LSP onde será encaminhado o fluxo de dados correspondente;
- a taxa de transmissão a ser configurada no l-ésimo ($1 \leq l \leq l_k$) nível do k-ésimo LSP é λ_k^l sendo esta medida em $Kbps$;
- o máximo atraso fim-a-fim permitido ao k-ésimo LSP é h_k sendo este medido em ms.
- o parâmetro δ_k é introduzido na função objetivo para indicar a importância do k-ésimo fluxo de dados, que será encaminhado no k-ésimo LSP, denominado

parâmetro de prioridade de admissão, o qual atua como elemento de differenciação de serviços.

- o parâmetro μ_{ij} dá a capacidade de transmissão do enlace (i, j).

O problema de engenharia de tráfego (PET) pode ser colocado em programação matemática:

$$\text{PET}: \text{ Maximize } \sum_{k=1}^{K}\sum_{l=1}^{l_k} \delta_k \lambda_k^l y_k^l \tag{9.1}$$

Sujeita a :

$$\sum_{l=1}^{l_k} y_k^l = 1 \qquad \forall k \in \mathcal{K} \tag{9.2}$$

$$\sum_{k=1}^{K}\sum_{l=1}^{l_k} \lambda_k^l x_{ij}^{kl} \leq \mu_{ij} \qquad \forall (i,j) \in E \tag{9.3}$$

$$\sum_{(i,j)\in E}\sum_{l=1}^{l_k} c_{ij} x_{ij}^{kl} \leq h_k \qquad \forall k \in \mathcal{K} \tag{9.4}$$

$$\sum_{\{j:(i,j)\in E\}} x_{ij}^{kl} - \sum_{\{j:(j,i)\in E\}} x_{ji}^{kl} = b_i^k y_k^l \qquad \forall i \in V, \forall k \in \mathcal{K}, \forall l \in L_k \tag{9.5}$$

$$x_{ij}^{kl} \in \{0,1\} \qquad \forall (i,j) \in E, \forall k \in \mathcal{K}, \forall l \in L_k \tag{9.6}$$

$$y_k^l \in \{0,1\} \qquad \forall k \in \mathcal{K}, \forall l \in L_k \tag{9.7}$$

onde: $b_i^k = 1$ se $i = s_k$, $b_i^k = -1$ se $i = d_k$, e $b_i^k = 0$ caso contrário. E ainda $\mathcal{K} = \{1, \ldots, K\}$ e $L_k = \{1, \ldots, l_k\}$.

A restrição (9.2) garante que cada LSP será configurado precisamente em um único nível de transmissão. A restrição (9.3) impõe os limites de largura de banda dos enlaces. A restrição (9.4) modela os limites máximos de atraso de cada LSP. A expressão (9.5) garante que cada LSP k será encaminhado em um caminho contínuo, não paralelo do nó origem s_k até o destino d_k. Finalmente as expressões (9.6) e (9.7) definem as restrições Booleanas para as variáveis de decisão.

9.4.1 Formulação em MOSEL

Abaixo apresentamos um modelo em linguagem Mosel para PET, mas omitimos detalhes das diretivas e truques de modelagem selecionados. O leitor poderá recorrer ao manual de usuário de Mosel e livros especializados [25] para maiores informações. Acreditamos que o modelo cobre elementos-chave da linguagem para modelagem avançada que invariavelmente serão necessários em aplicações teórico-práticas. Assumiremos que o modelo ficará armazenado em um arquivo tipo texto denominado “model.mos”.

```
!-------------------------------------------------------------
! Mosel formulation of the fully-fledged LSP-routing problem

model lsp
```

```
uses "mmxprs"

declarations
  Node: set of integer

  Arc: array(Node,Node) of integer
  Arc_u: array(Node,Node) of integer
  Arc_d: array(Node,Node) of integer

  Lsp: set of integer
  Level: set of integer
  Lsp_s: array(Lsp) of integer
  Lsp_d: array(Lsp) of integer
  Lsp_p: array(Lsp) of integer
  Lsp_delay: array(Lsp) of integer
  Lsp_lbd: array(Lsp,Level) of integer

  x: array(Lsp,Level,Node,Node) of mpvar
  y: array(Lsp,Level) of mpvar
end-declarations

! Read in data from external file
initializations from "model.dat"
  Node
  Arc Arc_u Arc_d
  Lsp Level
  Lsp_s Lsp_d Lsp_p Lsp_delay Lsp_lbd
end-initializations

! Printing data
!-------------------------------------------------
fopen("f_data.txt",F_OUTPUT)
writeln
writeln("List of nodes")
writeln("-----------------------------------")
forall(i in Node)
  writeln("node(",i,"): ",Node(i))

writeln
writeln("List of Arcs")
writeln("-----------------------------------")
forall(i in Node, j in Node | Arc(i,j)=1 )
  writeln("arc: (",i,",",j,") u:",Arc_u(i,j)," delay: ",Arc_d(i,j))

writeln
writeln("List of Lsp's")
writeln("-----------------------------------")
forall(k in Lsp)
  writeln("lsp: (",k,"): ",Lsp(k),
      " s: ",Lsp_s(k)," d: ",Lsp_d(k),
      " p: ",Lsp_p(k))

writeln
writeln("List of bounds for Lsp's")
```

```
writeln("-------------------------------------")
forall(k in Lsp) do
  write("lsp: ",k," lbd:")
  forall(l in Level)
    write(" ",Lsp_lbd(k,l))
  writeln
end-do

fclose(F_OUTPUT)

! Instantiate variables
forall(k in Lsp, l in Level, i in Node, j in Node | Arc(i,j)=1)
  create(x(k,l,i,j))
forall(k in Lsp, l in Level)
  create(y(k,l))

forall(k in Lsp, l in Level, i in Node, j in Node | Arc(i,j)=1)
  x(k,l,i,j) is_binary
forall(k in Lsp, l in Level)
  y(k,l) is_binary

! Objective Function
Total_Performance := sum(k in Lsp, l in Level)
    Lsp_p(k)*Lsp_lbd(k,l)*y(k,l)

! arc capacity
forall(i in Node, j in Node | Arc(i,j)=1)
  sum(k in Lsp, l in Level) Lsp_lbd(k,l)*x(k,l,i,j) <= Arc_u(i,j)

! maximum delay
forall(k in Lsp)
  sum(i in Node, j in Node, l in Level | Arc(i,j)=1)
      Arc_d(i,j)*x(k,l,i,j) <= Lsp_delay(k)

! Network Flow Constraints
forall(k in Lsp, l in Level, i in Node | i<>Lsp_s(k) and i<>Lsp_d(k) )
        sum(j in Node | Arc(i,j)=1) x(k,l,i,j)
      - sum(j in Node | Arc(j,i)=1) x(k,l,j,i) = 0

forall(k in Lsp, l in Level)
     sum(j in Node | Arc(Lsp_s(k),j)=1) x(k,l,Lsp_s(k),j)
   - sum(j in Node | Arc(j,Lsp_s(k))=1) x(k,l,j,Lsp_s(k)) = y(k,l)

forall(k in Lsp, l in Level)
     sum(j in Node | Arc(Lsp_d(k),j)=1) x(k,l,Lsp_d(k),j)
   - sum(j in Node | Arc(j,Lsp_d(k))=1) x(k,l,j,Lsp_d(k)) = -y(k,l)

forall(k in Lsp)
   sum(l in Level) y(k,l) = 1

maximize(Total_Performance)

exportprob(EP_MAX,"t1.txt",Total_Performance)

fopen("f_sol.txt",F_OUTPUT)
```

```
writeln
writeln('Solution Found')
writeln('------------------------')
forall(k in Lsp, l in Level)
  if( getsol(y(k,l))<>0) then
     writeln("y(",k,",",l,") = ",getsol(y(k,l)))
  end-if
writeln

forall(k in Lsp, l in Level, i in Node, j in Node | Arc(i,j)=1)
  if( getsol(x(k,l,i,j))<>0) then
     writeln("x(",k,",",l,",",i,",",j,") = ",getsol(x(k,l,i,j)))
  end-if
writeln
writeln("objective: ",getobjval)
fclose(F_OUTPUT)

end-model
```

Instância Exemplo

Abaixo segue o conteúdo do arquivo "model.dat" com os parâmetros da instância a ser carregado quando da execução do modelo "model.mos" dado acima. Observe que em "model.mos" é feita referência ao arquivo "model.dat". A rede da instância exemplo está ilustrada na Figura 9.1.

```
Lsp: [1 2 3 4 5 6 7 8 9 10
      11 12 13 14 15 16 17 18 19 20 21]

Node: [1 2 3 4 5 6 7]

Arc: [ (1 3) 1    (3 1) 1    (2 3) 1    (3 2) 1
       (3 4) 1    (4 3) 1    (3 5) 1    (5 3) 1
       (4 6) 1    (6 4) 1    (5 6) 1    (6 5) 1
       (6 7) 1    (7 6) 1]

Level: [0 1 2]

Arc_u: [(1 3) 100    (3 1) 100    (2 3) 100    (3 2) 100
        (3 4) 100    (4 3) 100    (3 5) 100    (5 3) 100
        (4 6) 100    (6 4) 100    (5 6) 100    (6 5) 100
        (6 7) 100    (7 6) 100]

Arc_d: [(1 3) 1    (3 1) 1    (2 3) 1    (3 2) 1
        (3 4) 1    (4 3) 1    (3 5) 1    (5 3) 1
        (4 6) 1    (6 4) 1    (5 6) 1    (6 5) 1
        (6 7) 1    (7 6) 1]

Lsp_s: [2 3 1 1 6 4 5 1 1 1 6 3 3 1 2 6 5 7 7 7 7]

Lsp_d: [5 1 4 5 5 2 2 2 2 6 2 1 6 4 4 2 4 5 3 2 1]

Lsp_p: [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
```

```
Lsp_delay: [20 20 20 20 20 20 20 20 20 20 20
            20 20 20 20 20 20 20 20 20 20]


Lsp_lbd: [(1 0) 0  (1 1) 8   (1 2) 16
          (2 0) 0  (2 1) 10  (2 2) 20
          (3 0) 0  (3 1) 14  (3 2) 28
          (4 0) 0  (4 1) 5   (4 2) 11
          (5 0) 0  (5 1) 1   (5 2) 2
          (6 0) 0  (6 1) 3   (6 2) 6
          (7 0) 0  (7 1) 8   (7 2) 17
          (8 0) 0  (8 1) 11  (8 2) 22
          (9 0) 0  (9 1) 9   (9 2) 18
          (10 0) 0 (10 1) 1  (10 2) 3
          (11 0) 0 (11 1) 2  (11 2) 5
          (12 0) 0 (12 1) 13 (12 2) 26
          (13 0) 0 (13 1) 1  (13 2) 2
          (14 0) 0 (14 1) 6  (14 2) 13
          (15 0) 0 (15 1) 14 (15 2) 28
          (16 0) 0 (16 1) 4  (16 2) 9
          (17 0) 0 (17 1) 5  (17 2) 10
          (18 0) 0 (18 1) 4  (18 2) 8
          (19 0) 0 (19 1) 5  (19 2) 10
          (20 0) 0 (20 1) 6  (20 2) 13
          (21 0) 0 (21 1) 2  (21 2) 4]
```

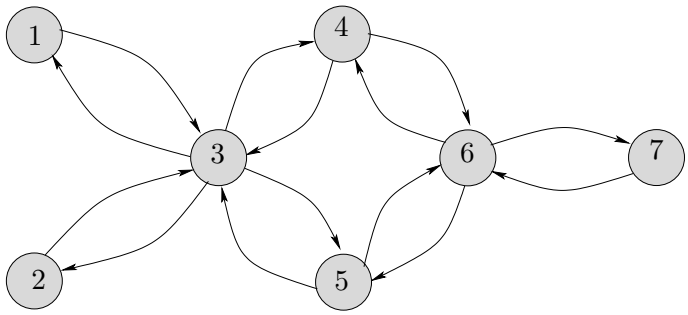


Figura 9.1: Rede exemplo do problema de alocação de rotas.

9.4.2 Formulação em AMPL

De forma similar à linguagem MOSEL, apresentamos abaixo um modelo AMPL do problema de alocação de rotas (arquivo "model.amp") e os dados da instância exemplo (arquivo "model.dat").

```
set LSPS;               # Set of LSP
set NODES;              # nodes of the communication network
set ARCS within (NODES cross NODES); # set of arcs
set LEVELS;             # various amounts for LSPs

param lk_u     {(i,j) in ARCS};    # arc capacity
param lk_d     {(i,j) in ARCS};    # arc delay
```

```
param lsp_lbd  {(k,l) in LSPS cross LEVELS}; # lbd_k^l
param lsp_p    {k in LSPS};        # lsp priority
param lsp_s    {k in LSPS};        # lsp source
param lsp_d    {k in LSPS};        # lsp destination
param lsp_delay {k in LSPS};       # lsp maximum delay transmission


#-------------------------------------------------------------
var x  { k in LSPS, l in LEVELS, (i,j) in ARCS} integer  >=0, <=1;
var y  { k in LSPS, l in LEVELS}  integer >=0, <=1;

#-------------------------------------------------------------
maximize total_effect:
      sum{k in LSPS, l in LEVELS}  lsp_p[k]*lsp_lbd[k,l]*y[k,l];

#-------------------------------------------------------------
subject to Lk_Bound {(i,j) in ARCS}:
    sum{k in LSPS, l in LEVELS} lsp_lbd[k,l]*x[k,l,i,j]
        <= lk_u[i,j];

#-------------------------------------------------------------
subject to Lsp_Delay {k in LSPS}:
   sum{ (i,j) in ARCS, l in LEVELS }
        lk_d[i,j]*x[k,l,i,j] <= lsp_delay[k];

#-------------------------------------------------------------
subject to NetFlow_0 {k in LSPS, l in LEVELS,
                      i in NODES diff {lsp_s[k],lsp_d[k]} } :
         sum{(i,j) in ARCS} x[k,l,i,j]
       - sum{(j,i) in ARCS} x[k,l,j,i] = 0;

#-------------------------------------------------------------
subject to NetFlow_s {k in LSPS, l in LEVELS } :
        sum{(lsp_s[k],j) in ARCS} x[k,l,lsp_s[k],j]
      - sum{(j,lsp_s[k]) in ARCS} x[k,l,j,lsp_s[k]] = 1*y[k,l];

#-------------------------------------------------------------
subject to NetFlow_d {k in LSPS, l in LEVELS } :
        sum{(lsp_d[k],j) in ARCS} x[k,l,lsp_d[k],j]
      - sum{(j,lsp_d[k]) in ARCS} x[k,l,j,lsp_d[k]] = -1*y[k,l];

#-------------------------------------------------------------
subject to Connection {k in LSPS} :
    sum{l in LEVELS} y[k,l] = 1;
```

Instância Exemplo

```
set LSPS := 1 2 3 4 5 6 7 8 9 10
            11 12 13 14 15 16 17 18 19 20 21;

set NODES := 1 2 3 4 5 6 7;

set ARCS :=  (1,3) (3,1) (2,3) (3,2)
             (3,4) (4,3) (3,5) (5,3)
             (4,6) (6,4) (5,6) (6,5)
```

```
            (6,7) (7,6);

set LEVELS := 0 1 2;

param:     lk_u  lk_d :=
          1 3 100 1
          3 1 100 1
          2 3 100 1
          3 2 100 1
          3 4 100 1
          4 3 100 1
          3 5 100 1
          5 3 100 1
          4 6 100 1
          6 4 100 1
          5 6 100 1
          6 5 100 1
          6 7 100 1
          7 6 100 1;

param: lsp_s    lsp_d   lsp_p lsp_delay :=
       1   2   5   1   20
       2   3   1   1   20
       3   1   4   1   20
       4   1   5   1   20
       5   6   5   1   20
       6   4   2   1   20
       7   5   2   1   20
       8   1   2   1   20
       9   1   2   1   20
       10  1   6   1   20
       11  6   2   1   20
       12  3   1   1   20
       13  3   6   1   20
       14  1   4   1   20
       15  2   4   1   20
       16  6   2   1   20
       17  5   4   1   20
       18  7   5   1   20
       19  7   3   1   20
       20  7   2   1   20
       21  7   1   1   20;

param lsp_lbd:
      0   1   2 :=
  1   0   8   16
  2   0   10  20
  3   0   14  28
  4   0   5   11
  5   0   1   2
  6   0   3   6
  7   0   8   17
  8   0   11  22
  9   0   9   18
  10  0   1   3
```

```
11  0   2   5
12  0   13  26
13  0   1   2
14  0   6   13
15  0   14  28
16  0   4   9
17  0   5   10
18  0   4   8
19  0   5   10
20  0   6   13
21  0   2   4;
```

Para agregar os dados ao modelo, realizar a otimização e, ao final, apresentar os resultados, se faz necessário colocar diretivas em um arquivo de comandos (arquivo "run.cmd"). O conteúdo do arquivo "run.cmd" segue abaixo.

```
solve;
display y;

display sum{k in LSPS, l in LEVELS}
  lsp_p[k]*lsp_lbd[k,l]*y[k,l];

for {k in LSPS, l in LEVELS} {
      display k;
      display l;
      display { (i,j) in ARCS } x[k,l,i,j];
}
```

9.5 Referências

Para informações detalhadas e exemplos de uso avançado da linguagem AMPL, recomendamos o livro de Fourer, Gay e Kernighan [20]. Este texto apresenta a linguagem por meio de exemplos que crescem em complexidade gradualmente sendo, portanto, recomendável a todas as classes de usuários. A leitura do livro é obrigatória para aqueles que desejam utilizar a linguagem na prática e em pesquisa.

Da mesma forma que AMPL, o texto de Guéret, Prins e Sevaux [25] é uma das referências mais completas da linguagem MOSEL. São dados exemplos que variam dos mais simples aos mais complexos juntamente com os respectivos modelos MOSEL. Estes podem servir de base para a construção de modelos adequados às necessidades do usuário.

9.6 Exercícios

EX 9.1: Um conjunto de n tarefas devem ser executadas em uma única máquina que pode executar apenas uma tarefa de cada vez. Cada tarefa j leva p_j horas para ser finalizada. Dados os pesos das tarefas, $w_j = 1, \ldots, n$, em que ordem as tarefas devem ser executadas de maneira a se minimizar a soma ponderada do início das tarefas?

1) Formule o problema de sequenciar as tarefas como um problema de programação matemática inteira mista. (Sugestão: use variáveis $x_{ij} = 1$ se a tarefa i precede j, e 0 caso contrário; e s_j como o início da tarefa j.)

2) Implemente o modelo desenvolvido em AMPL e/ou Xpress-MP. Encontre a solução ótima para a instância dada na Tabela 9.2

Tabela 9.2: Sequenciamento de tarefas

Tarefa j	Peso w_j	Tempo p_j
1	0.5	1.5
2	0.4	2.3
3	0.1	1.2
4	0.5	1.0
5	0.3	1.5
6	0.5	3.6
7	0.2	4.1
8	0.8	0.8
9	1.0	2.0
10	0.6	3.0

EX 9.2: Codifique em AMPL os exercícios EX 1.1, EX 1.3, EX 1.4, EX 1.5 e EX 1.6. Encontre as soluções executando os respectivos modelos no servidor NEOS (`http://www-neos.mcs.anl.gov/neos/server-solvers.html`).

Capítulo 10

Fundamentos de Programação Inteira

Este capítulo inicia o estudo da programação inteira. Após apresentarmos alguns problemas motivadores, descrevemos formalmente alguns casos particulares de problemas de programação inteira e apresentamos um roteiro para modelagem de problemas. Ao final introduzimos conceitos fundamentais para formulação eficaz de problemas, em particular formas de comparação e avaliação da qualidade de formulações alternativas para um problema.

O material a seguir é uma síntese do Capítulo 1 do livro de Wolsey [44].

10.1 Introdução

Uma variedade expressiva de problemas de cunho prático e teórico podem ser formulados em programação linear inteira, incluindo problemas combinatórios freqüentemente encontrados em teoria dos grafos, problemas de lógica, situações práticas compreendendo a logística de empresas, entre outros. No que segue, delineamos dois problemas práticos tipicamente resolvidos por meio de modelos e algoritmos de programação inteira.

10.1.1 Escalonamento de Trens

Considere uma empresa de transporte ferroviário que deseja encontrar um escalonamento de seus trens, ou seja, rotas incluindo datas de partida, chegadas e tempos de permanência em estações ao longo das rotas. Um aspecto relevante é o fato de que as rotas se repetem diariamente, justificando portanto a otimização dos escalonamento. Dados acerca do problema também estão disponíveis e devem ser levados em consideração: os tempos de percurso de uma estação para outra são conhecidos; dois trens não podem trafegar na mesma linha, a menos que separados por alguns minutos; e para facilitar conexões, o horário de partida de um trem A deve suceder o horário de chegada de um trem B por alguns minutos. Enfim, o problema é encontrar um escalonamento factível que minimize algum critério econômico.

10.1.2 *Airline Crew Scheduling*

O problema de alocar tripulações a vôos sujeito a restrições físicas, temporais, de segurança e trabalhistas constitui uma tarefa árdua de grande impacto econômico para empresas aéreas, tanto que a indústria de aviação comercial americana possui departamentos de pesquisa operacional responsáveis por problemas desta natureza. Dados o itinerário dos vôos para um tipo particular de avião, um problema é projetar a escala semanal das tripulações, onde cada tripulação deve ser designada para um período consistindo de um conjunto de vôos que satisfaça uma série de restrições. Dentre elas, ressaltamos: o número máximo de horas de vôo, os períodos de descanso mínimo, e a necessidade de que os vôos sejam cíclicos permitindo o retorno periódico das tripulações às suas respectivas bases. O problema é encontrar uma escala das tripulações que minimize o custo total, que é função do tempo de vôo, período de descanso e muitos outros fatores.

10.2 O Que É um Problema Inteiro?

Em palavras, um problema inteiro se refere ao problema de encontrar um vetor de inteiros que minimize uma função linear deste vetor, ao mesmo tempo que restrições lineares são satisfeitas. Formalmente, o problema de programação linear pode ser expresso em programação matemática como segue:

$$PL: \qquad Max\left\{c^T x : Ax \leqslant b, x \geqslant 0\right\}$$

onde $x \in \mathbb{R}^{n\times 1}$, $A \in \mathbb{R}^{m\times n}$, $c \in \mathbb{R}^{n\times 1}$ e $b \in \mathbb{R}^{m\times 1}$. O problema geral de programação inteira pode ser divido em classes, dependendo da natureza das decisões, da existência de variáveis contínuas e discretas e do tipo de restrições. Abaixo apresentamos algumas dessas classes.

10.2.1 Problema (Linear) Inteiro Misto

O problema inteiro misto nada mais é do que um problema de programação linear com variáveis discretas e outras contínuas, sendo expresso na forma:

$$\begin{array}{lll} PIM: & Max & c^T x + h^T y \\ & s.a: & Ax + Gy \leqslant b \\ & & x \geqslant 0,\ y \geqslant 0 \text{ e } y \text{ inteiro.} \end{array}$$

10.2.2 Problema (Linear) Inteiro

Nesta versão do problema, todas as variáveis devem ser inteiras:

$$\begin{array}{lll} PI: & Max & c^T x \\ & s.a: & Ax \leqslant b \\ & & x \in \mathbb{Z}^n_+ \end{array}$$

10.2.3 Problema Linear Binário

Uma versão particular do problema inteiro é o problema binário, no qual todas as variáveis só podem assumir os valores 0 ou 1:

$$\begin{array}{rll} PIB: & Max & c^T x \\ & s.a: & Ax \leqslant b \\ & & x \in \{0,1\}^n \end{array}$$

A relevância de PIB surge do desenvolvimento de resultados teóricos e algoritmos específicos para esta classe de problema. Um ponto interessante é o fato que um PI pode ser reduzido em um problema equivalente do tipo PIB.

10.2.4 Problema de Otimização Combinatória

Problemas de natureza combinatória compreendem aqueles em que se deseja encontrar um arranjo particular, dentre um número comumente combinatório e exponencial de arranjos factíveis, que minimize/maximize algum critério de seleção para os diferentes arranjos. Seja $N = \{1, \ldots, n\}$ um conjunto finito, c_j o custo de cada elemento j de N, e $\mathcal{F}$ uma família de subconjuntos de N dita família de subconjuntos viáveis, e.g., $\mathcal{F} \subseteq 2^N$. Então o problema de otimização combinatória pode ser definido em programação matemática:

$$POC: \; Max_{S \subseteq N} \left\{ \sum_{j \in S} c_j \; : \; S \in \mathcal{F} \right\}$$

Freqüentemente, POC pode ser formulado como PI ou PIB.

10.3 Programação Linear e Arredondamento

Por que não utilizar PL? Poderíamos desconsiderar as restrições de variáveis inteiras, obter uma solução ótima x^* para PL e depois arredondar x^* de forma a se obter uma solução para PI, por exemplo. Infelizmente, esta abordagem não funciona como mostra o contra-exemplo abaixo:

$$\begin{array}{rrcrcr} Max & x_1 & + & 0.6x_2 & & \\ s.a: & 50x_1 & + & 31x_2 & \leqslant & 250 \\ & 3x_1 & - & 2x_2 & \geqslant & -4 \end{array}$$

$$x_1,\, x_2 \geqslant 0 \text{ e inteiros}$$

A solução ótima para PL, $x_{PL} = (\frac{376}{193}, \frac{950}{193}) = (1.94, 4.92)$, poderia ser arredondada para a solução $\overline{x}_{PL} = (2, 4)$, que é bastante "distante" da solução ótima $x^* = (5, 0)$. Uma ilustração é dada na Figura 10.1.

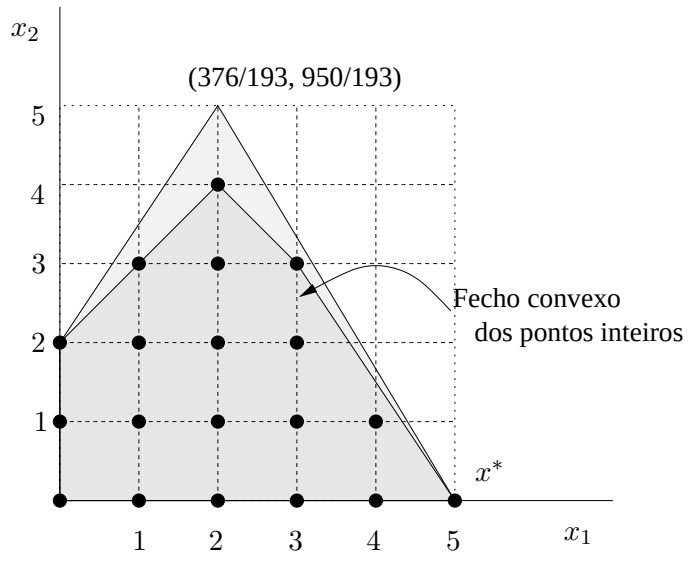


Figura 10.1: Ilustração da solução obtida através de arredondamento

10.4 Formulação de PIs e PIBs

A formulação de um problema é tanto uma arte quanto uma ciência. Existem técnicas e metodologias, contudo a experiência e a intuição continuam e provavelmente continuarão a desempenhar um papel fundamental. Aqui apresentamos alguns passos metodológicos para formulação de problemas que são exemplificados. A transcrição de um problema em uma formulação deve ser conduzida de uma forma sistemática, mantendo sempre claro a distinção entre os *dados do problema* e as *variáveis de decisão*. Passos sugeridos:

i. Defina quais variáveis são necessárias

ii. Utilize as variáveis para definir restrições tal que os pontos factíveis correspondam às soluções viáveis da formulação

iii. Utilize as variáveis para definir a função objetivo

Se as dificuldades persistirem, defina conjuntos de variáveis adicionais alternativas, repetindo os passos metodológicos.

10.4.1 Exemplo 1: Formulando o Problema de Alocação

No problema de alocação, há n pessoas para executar n tarefas, sendo cada pessoa obrigatoriamente designada a precisamente a uma tarefa. Alguns indivíduos são menos eficientes em certas tarefas, incorrendo um custo c_{ij} se a pessoa i executar a tarefa j. Portanto o problema é encontrar a alocação pessoa-tarefa de menor custo agregado.

Definindo as Variáveis

$$x_{ij} = \begin{cases} 1 & \text{se a pessoa } i \text{ é alocada à tarefa } j \\ 0 & \text{caso contrário} \end{cases}$$

Definindo as Restrições

a) Cada pessoa i executa uma tarefa

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad i = 1, \ldots, n$$

b) Cada tarefa j é executada por uma pessoa

$$\sum_{i=1}^{n} x_{ij} = 1 \qquad j = 1, \ldots, n$$

c) As variáveis são 0-1

$$x_{ij} \in \{0, 1\} \qquad i = 1, \ldots, n \quad j = 1, \ldots, n$$

Definindo a Função Objetivo

$$\text{Min} \quad \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}$$

10.4.2 Exemplo 2: O Problema da Mochila

O problema da mochila é chave em várias aplicações teórico-práticas. Um exemplo é o problema enfrentado por um investidor que dispõe de uma quantidade b para investimentos. Há n projetos sob consideração, sendo a_j o custo estimado do projeto j e c_j é o retorno esperado. O objetivo do investidor se resume a encontrar um conjunto de projetos tal que a quantidade de investimento não exceda a quantidade disponível e, ao mesmo tempo, maximize o retorno.

Definindo as Variáveis

$$x_j = \begin{cases} 1 & \text{se o projeto } j \text{ é selecionado} \\ 0 & \text{caso contrário} \end{cases}$$

Definição de Restrições

a) Limitação de recursos: $\sum_{j=1}^{n} a_j x_j \leqslant b$

b) Variáveis são 0-1: $x_j \in \{0, 1\}$ $j = 1, \ldots, n$

Definindo a Função Objetivo

$$Max \sum_{j=1}^{n} c_j x_j$$

10.4.3 Exemplo 3: O Problema de Cobertura por Conjuntos

O problema de cobertura por conjuntos define outra classe empregada com freqüência na prática. Para se ter uma idéia, o problema de escalonamento de tripulações pode ser resolvido por meio da decomposição em sub-problemas, um dos quais é um problema de cobertura por conjuntos, onde os elementos do conjunto base (S) são vôos a serem operados pela empresa de transporte aéreo e os sub-conjuntos ($S_j \subset S$) correspondem a configurações de tripulações viáveis. Aqui vamos considerar o caso particular de localização de unidades de emergência. Dado um certo número de regiões, o problema é decidir onde devemos instalar um conjunto de serviços de emergência. Para cada ponto possível de instalação de um centro, sabemos o custo de instalação e as regiões que a central de serviços pode atender (e.g., em menos de 8 minutos) Seja $M = \{1, \ldots, m\}$ o conjunto de regiões e $N = \{1, \ldots, n\}$ o conjunto de possíveis centrais de emergência. Seja ainda $S_j \subseteq M$ o conjunto de regiões que podem ser atendidas pela central j e c_j o custo de instalação. Então, obtemos o seguinte problema de otimização combinatória:

$$POC: \ Min_{T \subseteq N}\{\sum_{j \in T} c_j : \bigcup_{j \in T} S_j = M\}$$

Formulando o Problema Como um PIB

Primeiro construímos a matriz A tal que $a_{ij} = 1$ se $i \in S_j$ e $a_{ij} = 0$ caso contrário, portanto $A \in \{0,1\}^{m \times n}$.

Definindo as Variáveis

$$x_j = \begin{cases} 1 & \text{se o central } j \text{ será instalada} \\ 0 & \text{caso contrário} \end{cases}$$

Definindo as Restrições

a) Pelo menos um centro deve servir a região i

$$\sum_{j=1}^{n} a_{ij} x_j \geqslant 1 \qquad \text{para } i = 1, \ldots, m$$

b) As variáveis são binárias

$$x_j \in \{0,1\} \qquad \text{para } j = 1, \ldots, n$$

Definindo a Função Objetivo

$$\text{Min} \quad \sum_{j=1}^{n} c_j x_j$$

10.4.4 Exemplo 4: O Problema do Caixeiro Viajante (PCV)

Outro problema clássico é o problema do caixeiro viajante, o qual consiste em escolher uma ordem para um viajante partir da sua cidade base, digamos cidade 1, visitar as demais $n-1$ cidades precisamente uma vez, e depois retornar à cidade base de maneira que a distância percorrida seja a menor possível. Nos é dado um conjunto de n cidades e o custo (distância) c_{ij} do deslocamento da cidade i para a cidade j. A tarefa é encontrar uma rota (circuito) que visite cada cidade precisamente uma vez que seja de menor custo possível. As aplicações são diversar, em particular o roteamento de veículos, a soldagem de pontos de contato em placas de circuitos integrados, e a coleta de lixo sistematizada.

Definindo as Variáveis

$$x_{ij} = \begin{cases} 1 & \text{se o viajante se desloca da cidade } i \text{ para a cidade } j \\ 0 & \text{caso contrário} \end{cases}$$

Definindo as Restrições

a) O viajante deixa a cidade i exatamente uma vez: $\sum_{j=1}^{n} x_{ij} = 1 \qquad i = 1, \ldots, n$

b) O viajante entra na cidade j exatamente uma vez: $\sum_{i=1}^{n} x_{ij} = 1 \qquad j = 1, \ldots, n$

c) Restrições de conectividade:

$$\sum_{i \in S} \sum_{j \notin S} x_{ij} \geqslant 1 \qquad \forall S \subset N,\ S \neq \emptyset$$

ou eliminação de subrotas

$$\sum_{i \in S} \sum_{j \in S,\, j \neq i} x_{ij} \leqslant |S| - 1 \qquad \forall S \subseteq N,\ 2 \leqslant |S| \leqslant n-1$$

A Figura 10.2 ilustra a restrição de conectividade.

Definindo a função objetivo

$$\text{Min} \quad \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}$$

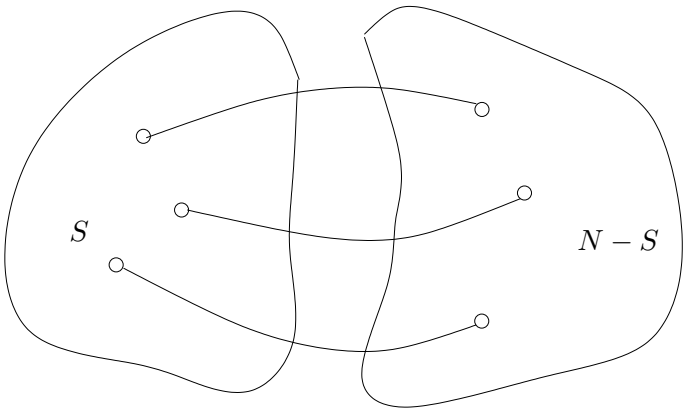


Figura 10.2: Ilustração da restrição de conectividade

10.5 Explosão Combinatória

Todos os problemas até então vistos são combinatoriais: a solução é um subconjunto de um conjunto finito. Em princípio, tendo em vista que o conjunto de soluções factíveis é finito, poderíamos enumerar todas as soluções factíveis. As Tabelas 10.1 e 10.2 ilustram o crescimento do número de soluções factíveis para alguns problemas. A partir desta análise preliminar, podemos concluir que uma abordagem baseada em enumeração pode ser eficaz apenas em instâncias pequenas.

Tabela 10.1: Número de soluções em função do tamanho do problema

Tipo	Problema	Número de soluções
1	Alocação (10.4.1)	$n!$
2	Mochila e cobertura por conjuntos (10.4.2) e (10.4.3)	2^n
3	Problema do caixeiro (10.4.4) viajante assimétrico	$(n-1)!$

Tabela 10.2: Crescimento de funções

n	$log_2 n$	$\sqrt{n}$	n^2	2^n	$n!$
10	3.32	3.16	10^2	10^3	3.6×10^6
10^2	6.64	10	10^4	10^{30}	9.3×10^{157}
10^3	9.97	31.62	10^6	10^{301}	4×10^{2567}

10.6 Formulação de Problemas Inteiros Mistos (PIMS)

Modelos contendo variáveis discretas e contínuas apresentam propriedades sui generis, necessitando tratamento particular. Por exemplo, nos próximos capítulos veremos os cortes de Gomory que são diferenciados dependendo do problema ser inteiro ou inteiro misto. No que segue será apresentado um conjunto de truques de modelagem e exemplos que podem ser úteis.

10.6.1 Exemplo 1: Modelando Custos Fixos

Imagine a situação onde se deseja modelar uma função não-linear dada por:

$$h(x) = \begin{cases} f + px & \text{se } 0 < x \leqslant c \\ 0 & \text{se } x = 0 \end{cases}$$

A função $h(x)$ é ilustrada na Figura 10.3.

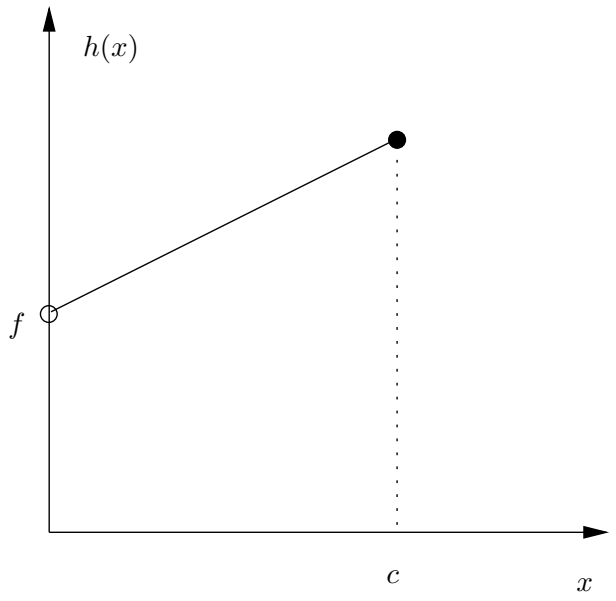


Figura 10.3: Função com custo fixo

Definição de uma variável adicional (y)

$$y = \begin{cases} 1 & \text{se } x > 0 \\ 0 & \text{se } x = 0 \end{cases}$$

Introduzindo a função objetivo e restrições

$$h(x) = fy + px$$
$$x \leqslant cy$$
$$y \in \{0, 1\}$$
$$x \geqslant 0$$

O modelo matemático acima é válido apenas para o caso de minimização. Note também que $x = 0$ e $y = 1$ podem ocorrer.

10.6.2 Exemplo 2: Localização de Depósitos sem Limites de Capacidade

Mais um exemplo de problema clássico que se preocupa com a localização de depósitos, e conseqüentemente o custo de instalação deste bem como a posição estratégica dos mesmos, além dos custos de transporte incorridos no transporte de mercadorias dos depósitos aos clientes. Nos é dado um conjunto $N = \{1, \ldots, n\}$ de possíveis localizações de depósitos, um conjunto $M = \{1, \ldots, m\}$ de clientes, o custo fixo f_j para a instalação do depósito j, e custo de transporte c_{ij} caso o suprimento da demanda do cliente i seja realizado pelo depósito j. Assim o problema consiste em decidir que depósitos serão instalados e quais depósitos atenderão quais clientes, de maneira a minimizar o custo total de instalação e transporte.

Definindo as variáveis

$$y_j = \begin{cases} 1 & \text{se o depósito } j \text{ será construído} \\ 0 & \text{caso contrário} \end{cases}$$

x_{ij} é a fração da demanda do cliente i suprida pelo depósito j

Definindo as restrições

a) Depósito j pode operar somente se instalado: $\sum_{i=1}^{m} x_{ij} \leqslant my_j \qquad j = 1, \ldots, n$

b) A demanda do cliente i deve ser satisfeita: $\sum_{j=1}^{n} x_{ij} = 1 \qquad i = 1, \ldots, m$

c) Variáveis 0-1:

$$y_j \in \{0, 1\} \qquad j = 1, \ldots, n$$
$$0 \leqslant x_{ij} \leqslant 1 \qquad i = 1, \ldots, m \text{ e } j = 1, \ldots, n$$

Especificando a função objetivo

$$\text{Min} \quad \sum_{j=1}^{n} f_j y_j + \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij}$$

Problema completo

$$
\begin{array}{ll}
\text{Minimize} & \sum_{j=1}^{n} f_j y_j + \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij} x_{ij} \\
\text{Sujeito a :} & \\
& \sum_{j=1}^{n} x_{ij} = 1 \qquad\qquad i = 1, \ldots, m \\
& \sum_{i=1}^{m} x_{ij} \leqslant m y_j \qquad\qquad j = 1, \ldots, n \\
& y_j \in \{0, 1\} \qquad\qquad j = 1, \ldots, n \\
& 0 \leqslant x_{ij} \leqslant 1 \qquad\qquad i = 1, \ldots, m \text{ e } j = 1, \ldots, n
\end{array}
\tag{10.1}
$$

10.6.3 Alternativas Discretas e Disjuntas

Uma área que tem recebido atenção de pesquisadores e está tendo resultados promissores na prática é a programação disjuntiva, ou seja, modelagem e algoritmos com base em disjunções. Para entender a programação disjuntiva, suponha que $x \in \mathbb{R}^n$ satisfaz

$$
\begin{array}{c}
0 \leqslant x \leqslant u \text{ e} \\
a_1^T x \leqslant b_1 \text{ ou } a_2^T x \leqslant b_2
\end{array}
\tag{10.2}
$$

Em outras palavras, x deve satisfazer uma ou outra restrição linear, não sendo necessário mas possível que ele satisfaça ambas as restrições. A região factível de uma disjunção com duas restrições é ilustrada na Figura 10.1. Note que a região factível não é convexa.

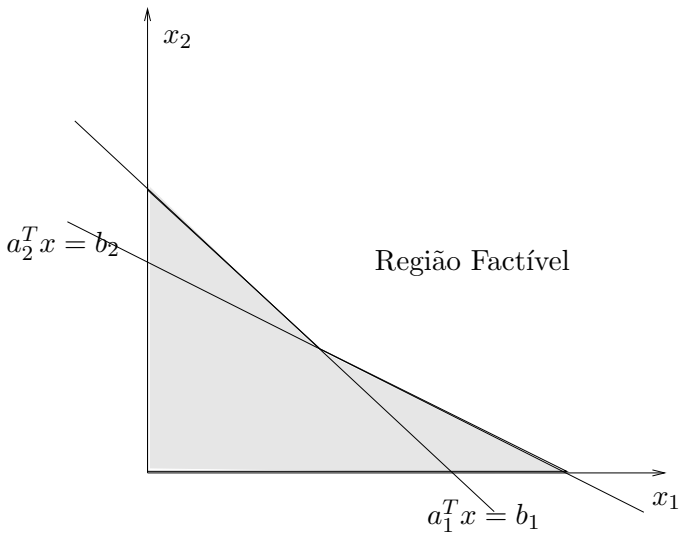


Figura 10.4: Região factível não convexa

Podemos representar as equações em (10.2), ou melhor, a *disjunção ou* em programação inteira mista? Sim é possível. Seja $M = \max_{i=1,2}\{a_i^T x - b_i : 0 \leqslant x \leqslant u\}$.

Primeiramente, introduzimos duas variáveis binárias, y_1 e y_2, cuja semântica é explicitada abaixo:

$$y_1 = \begin{cases} 1 & \text{se } x \text{ satisfaz } a_1^T x \leqslant b_1 \\ 0 & \text{caso contrário} \end{cases}$$

$$y_2 = \begin{cases} 1 & \text{se } x \text{ satisfaz } a_2^T x \leqslant b_2 \\ 0 & \text{caso contrário} \end{cases}$$

Uma vez introduzidas as variáveis acima, podemos apresentar a formulação completa:

$$\begin{gathered} a_1^T x \leqslant b_1 + M(1 - y_1) \\ a_2^T x \leqslant b_2 + M(1 - y_2) \\ y_1 + y_2 = 1 \\ y_1, y_2 \in \{0, 1\} \\ 0 \leqslant x \leqslant u \end{gathered}$$

Disjunções aparecem naturalmente em problemas de escalonamento. Suponha que as tarefas 1 e 2 devem ser processadas em uma mesma máquina, mas não simultaneamente. Seja p_i o tempo de processamento da tarefa i e t_i o instante em que o processamento é iniciado. Então, podemos expressar a precedência temporal de uma tarefa em relação a outra por meio de uma disjunção:

$$t_1 + p_1 \leqslant t_2 \text{ ou } t_2 + p_2 \leqslant t_1$$

10.7 Formulações Alternativas

Nesta e na próxima seção tentaremos entender o que leva uma formulação a ser melhor do que outra. Há situações em que uma formulação é superior a outra, em especial se leva a uma solução algorítmica mais eficiente.

Definição 10.1 *Um subconjunto do $\mathbb{R}^n$ descrito por um conjunto finito de restrições lineares $P = \{x \in \mathbb{R}^n : Ax \leqslant b\}$ é dito um poliedro.*

Definição 10.2 *Um poliedro $P \subseteq \mathbb{R}^{n+p}$ é uma formulação para um conjunto $X \subseteq \mathbb{Z}^n \times \mathbb{R}^p$ se, e somente se, $X = P \cap (\mathbb{Z}^n \times \mathbb{R}^p)$.*

Na Figura 10.5 são ilustradas duas formulações equivalentes para o conjunto $X = \{(1,1), (2,1), (3,1), (1,2), (2,2), (3,2), (2,3)\}$.

10.7.1 Formulações Equivalentes para o Problema da Mochila

Considere uma instância do problema da mochila com $n = 4$, onde o conjunto de soluções factíveis é $X = \{(0,0,0,0), (1,0,0,0), (0,1,0,0), (0,0,1,0), (0,0,0,1), (0,1,0,1), (0,0,1,1)\}$. Os poliedros abaixo induzem formulações válidas para o conjunto X:

$$\begin{aligned}
P_1 &= \{x \in \mathbb{R}^4 :\ 0 \leqslant x \leqslant 1,\ 83x_1 + 61x_2 + 49x_3 + 20x_4 \leqslant 100\} \\
P_2 &= \{x \in \mathbb{R}^4 :\ 0 \leqslant x \leqslant 1,\ 4x_1 + 3x_2 + 2x_3 + x_4 \leqslant 4\} \\
P_3 &= \{x \in \mathbb{R}^4 :\ 0 \leqslant x \leqslant 1,\ 4x_1 + 3x_2 + 2x_3 + x_4 \leqslant 4\,, \\
&\qquad\qquad x_1 + x_2 + x_3 \leqslant 1\,, \\
&\qquad\qquad x_1 + x_4 \leqslant 1\}
\end{aligned}$$

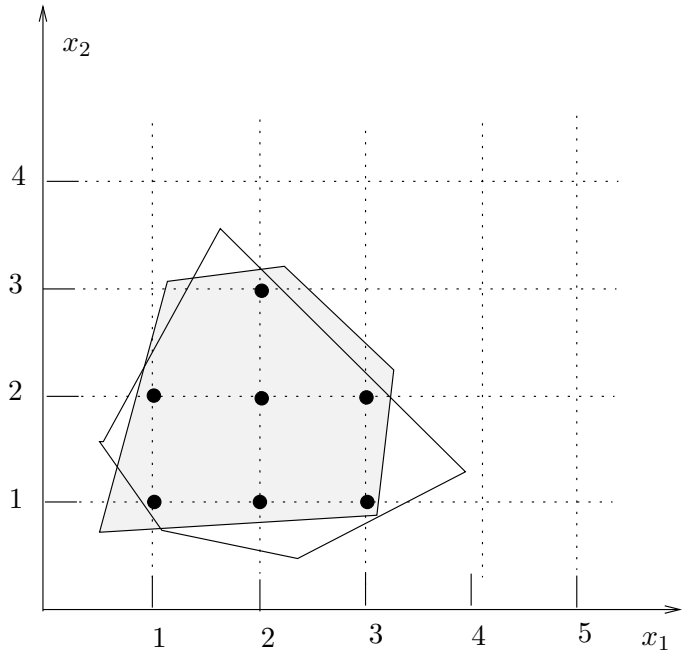


Figura 10.5: Formulações equivalentes alternativas

10.7.2 Localização de Depósitos sem Limites de Capacidade

Relembrando o problema de localização de depósitos sem capacidade, considere um depósito j e a restrição

$$\sum_{i=1}^{m} x_{ij} \leqslant m y_j, \quad y_j \in \{0,1\}, \quad 0 \leqslant x_{ij} \leqslant 1 \text{ para } i \in M \tag{10.3}$$

Logicamente, estas restrições expressam a condição: se $x_{ij} > 0$ então $y_j = 1$. Podemos então utilizar o seguinte conjunto de restrições alternativas:

$$\begin{array}{l} 0 \leqslant x_{ij} \leqslant y_j \quad \text{para } i \in M \\ y_j \in \{0,1\} \end{array} \tag{10.4}$$

Isto nos leva à formulação alternativa:

$$\begin{array}{lll} \text{Min} & \displaystyle\sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij} + \sum_{j=1}^{n} f_j y_j & \\ \text{S.a :} & \displaystyle\sum_{j=1}^{n} x_{ij} = 1 & i = 1,\ldots,m \\ & 0 \leqslant x_{ij} \leqslant y_j & i = 1,\ldots,m \text{ e } j = 1,\ldots,n \\ & y_j \in \{0,1\} & j = 1,\ldots,n \end{array} \tag{10.5}$$

Qual das duas formulações do problema de localização de facilidades é melhor, a formulação (10.1) ou a formulação (10.5)?

10.8 Formulações Apertadas e Ideais

Já observamos que o número de formulações válidas pode ser infinito. *Dentre um número potencialmente infinito de formulações, qual é a melhor formulação?* Por exemplo, considere o conjunto e as formulações equivalentes ilustrados na Figura 10.6.

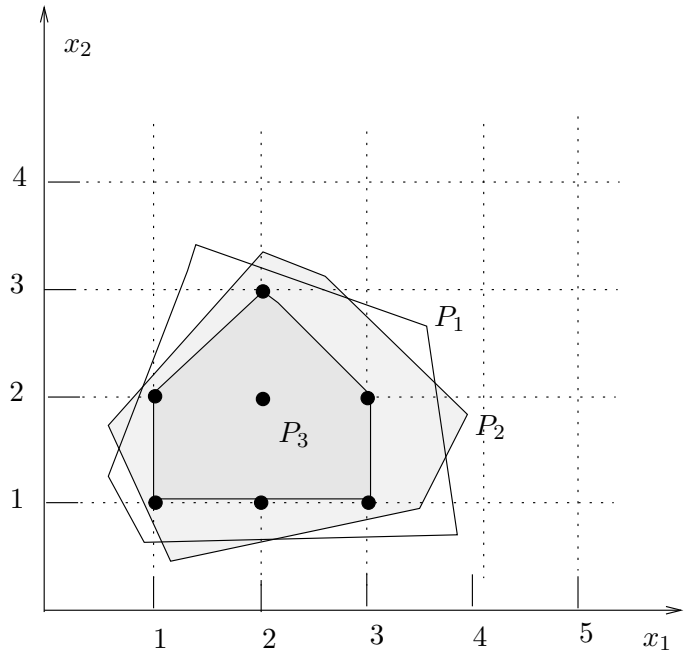


Figura 10.6: Formulações alternativas

Conforme a figura 10.6, a formulação P_3 é ideal: podemos resolver o PL usando P_3 (sem restrições inteiras) e sempre obtermos um ponto extremo inteiro; portanto, com P_3 o problema pode ser resolvido em tempo polinomial.

Definição 10.3 *Dado um conjunto $X \subseteq \mathbb{R}^n$, o fecho convexo de X, denotado por $conv(X)$ é definido como:*

$$conv(X) = \{x : x \text{ é combinação convexa dos elementos de } X\}$$

Proposição 10.1 *$conv(X)$ é um poliedro.*

Proposição 10.2 *Todos os pontos extremos de $conv(X)$ pertencem a X.*

Em princípio, podemos substituir o problema $\{Max\ c^T x : x \in X\}$ por $\{Max\ c^T x : x \in conv(X)\}$. *Então, qual é a dificuldade prática em se realizar esta substituição?* Tipicamente, necessitamos de um número exponencial de desigualdades para descrever o poliedro de $conv(X)$ e, além disso, não existe uma caracterização simples dos mesmos.

Definição 10.4 *Dado um conjunto $X \subseteq \mathbb{R}^n$ e duas formulações, P_1 e P_2 para X, dizemos que P_1 é melhor (mais apertada) do que P_2 se $P_1 \subset P_2$.*

10.8.1 Formulações Equivalentes para o Problema da Mochila

Vamos relembrar as três formulações alternativas para o problema da mochila sugerido na Seção 10.7.1.

$$
\begin{array}{lrcrcrcrcrl}
P_1 = \{x \in \mathbb{R}^4 : 0 \leqslant x \leqslant 1, & 83x_1 & + & 61x_2 & + & 49x_3 & + & 20x_4 & \leqslant & 100 & \} \\
P_2 = \{x \in \mathbb{R}^4 : 0 \leqslant x \leqslant 1, & 4x_1 & + & 3x_2 & + & 2x_3 & + & x_4 & \leqslant & 4 & \} \\
P_3 = \{x \in \mathbb{R}^4 : 0 \leqslant x \leqslant 1, & 4x_1 & + & 3x_2 & + & 2x_3 & + & x_4 & \leqslant & 4 & \\
& x_1 & + & x_2 & + & x_3 & & & \leqslant & 1 & \\
& x_1 & & & & & + & x_4 & \leqslant & 1 & \}
\end{array}
$$

Podemos verificar que $P_3 \subset P_2 \subset P_1$. Primeiro vamos mostrar que $P_2 \subset P_1$:

$$
\begin{gathered}
25(4x_1 + 3x_2 + 2x_3 + x_4) \leqslant 25 \times 4 \Rightarrow \\
100x_1 + 75x_2 + 50x_3 + 25x_4 \leqslant 100 \Rightarrow \\
83x_1 + 61x_2 + 49x_3 + 20x_4 \leqslant 100x_1 + 75x_2 + 50x_3 + 25x_4 \leqslant 100
\end{gathered}
$$

Portanto $x \in P_1 \nRightarrow x \in P_2$.

Agora mostraremos que $P_3 \subset P_2$: claramente, $P_3 \subseteq P_2$; o ponto $x = (\frac{1}{2}, 0, 1, 0) \in P_2$ mas $x \notin P_3$. As relações de inclusão entre as três formulações alternativas são ilustradas na Figura 10.7

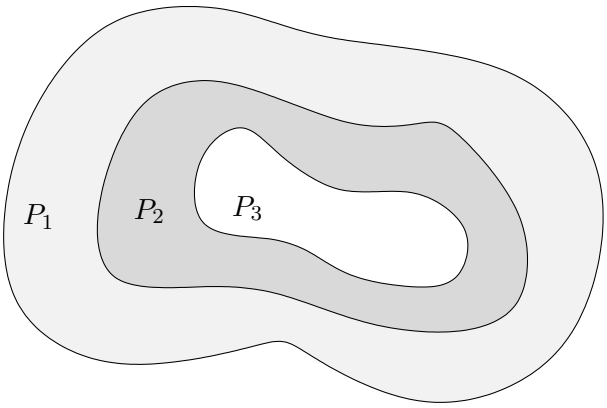


Figura 10.7: Formulações apertadas

10.8.2 Formulação para o Problema de Localização de Depósitos

As duas formulações só diferem quanto às restrições que permitem suprimento de clientes somente se um depósito estiver instalado. Relembrando as formulações.

$$P_1 : \qquad \sum_{i \in M} x_{ij} \leqslant m y_j \tag{10.6}$$

$$P_2 : x_{ij} \leqslant y_j \text{ para } i \in M \tag{10.7}$$

Claramente, (10.7) $\Rightarrow$ (10.6) e portanto $P_2 \subseteq P_1$. Considere o ponto x definido como:

$$
\begin{gathered}
x_{1j} = 1, x_{2j} = x_{3j} = \ldots = x_{mj} = 0 \\
y_j = \tfrac{1}{m}
\end{gathered}
$$

Verifica-se que o ponto $x \in P_1$, mas $x \notin P_2$

10.9 Referências

O texto apresentado neste capítulo, incluindo estrutura, conteúdo e figuras, foi traduzido do livro de Wolsey [44]. Planejamos estender o texto de forma a fazer uma contribuição original, através de outros exemplos práticos, formulações lineares por partes e formulações de problemas não lineares.

10.10 Exercícios

EX 10.1: Suponha que um investidor deseja investir em um conjunto $S = \{1, \ldots, 7\}$ de ações. Usando variáveis 0-1 formule as seguintes restrições:

i. não pode-se investir em todas as ações;

ii. pelo menos uma ação deve ser selecionada;

iii. a ação 1 não pode ser escolhida se a ação 3 é escolhida;

iv. a ação 4 pode ser escolhida apenas se a ação 2 é escolhida;

v. ambas as ações 1 e 5 devem ser escolhidas ou, exclusivamente, nenhuma delas; e

vi. pelo menos uma das ações do conjunto $\{1, 2, 3\}$ ou pelo menos duas ações do conjunto $\{2, 4, 5, 6\}$.

EX 10.2: Formule os seguintes problemas em programação inteira mista:

i. $u = min\{x_1, x_2\}$, assumindo que $0 \leqslant x_j \leqslant C$ para $j = 1, 2$;

ii. $v = |x_1 - x_2|$ com $0 \leqslant x_j \leqslant C$ para $j = 1, 2$; e

iii. o conjunto $X - \{x^*\}$ onde $X = \{x \in Z^n : Ax \leqslant b\}$ e $x^* \in X$.

EX 10.3: Sejam:

$$
\begin{aligned}
X_1 &= \{x \in \mathbb{B}^4 : 97x_1 + 32x_2 + 25x_3 + 20x_4 \leqslant 139\} \\
X_2 &= \{x \in \mathbb{B}^4 : 2x_1 + x_2 + x_3 + x_4 \leqslant 3\} \\
X_3 &= \{x \in \mathbb{B}^4 : \begin{array}{l} x_1 + x_2 + x_3 \leqslant 2 \\ x_1 + x_2 + x_4 \leqslant 2 \\ x_1 + x_3 + x_4 \leqslant 2\} \end{array}
\end{aligned}
$$

onde $\mathbb{B} = \{0, 1\}$. Mostre que $X_1 = X_2 = X_3$.

EX 10.4: Considere a instância do problema do caixeiro viajante simétrico conforme Figura 10.8. As arestas omissas possuem um custo proibitivo. Encontre uma solução ótima utilizando um pacote de software capaz de resolver problemas de programação inteira mista. (Sugestão utilize o software `lp_solve` instalado no diretório: `/home/pesquisa/camponog/lp_solve_1.4`.)

EX 10.5: Considere cada uma das relações lógicas envolvendo variáveis 0-1. Expresse estas relações como restrições lineares, possivelmente através do uso de variáveis 0-1 adicionais, da forma *mais apertada* possível:

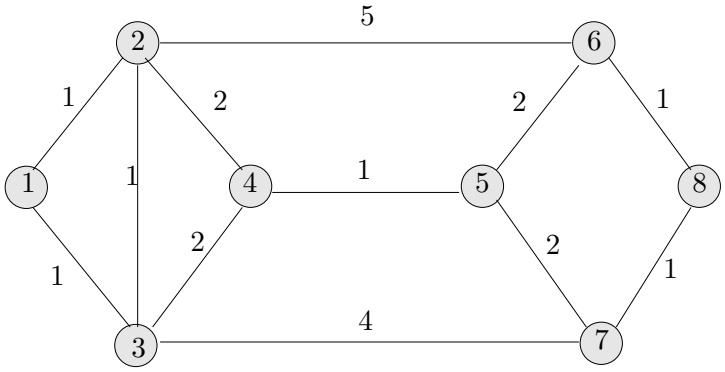


Figura 10.8: Grafo da instância do problema do caixeiro viajante

1) y_{ij} é igual a $x_i x_j$
2) Se $x_1 = 1$ então $x_2 + x_3 = 1$
3) x_3 é 1 se e somente se x_1 é 1 ou x_2 é 1, mas não ambas.
4) x_1 e x_2 podem assumir todos os valores 0-1 possíveis, exceto $x_1 = 0$ e $x_2 = 1$.

EX 10.6: Seja $a = (a_0, \ldots, a_n)$ uma seqüência de dígitos binários como indicado. Por exemplo, $a = (001111011010)$ é uma seqüência com $n = 11$.

i. Como você modelaria em programação matemática o problema de calcular x_j, $j = 0, \ldots, n$, onde $x_j = j - k$ é o comprimento da subsequência $(a_k, a_{k+1}, \ldots, a_{j-1})$ mais longa de 1's que termina em a_{j-1}? Defina $x_0 = 0$. Note que se $a_{j-1} = 0$ então $x_j = 0$.

ii. Para esta mesma questão, suponha que você deseja impor uma restrição de que a subsequência mais longa de $1's$ seja no máximo l. Como você expressaria esta restrição em programação matemática?

iii. Para esta mesma questão, suponha que você deseja impor a restrição de que cada subsequência de $1's$ tenha comprimento mínimo l_{min}, exceto ser for uma subsequência no extremo direito de a. Como você expressaria esta restrição em programação matemática?

EX 10.7: Uma empresa de distribuição de pacotes tem um conjunto de caminhões, motocicletas e outros veículos para entrega de pacotes. Seja M o número de pacotes a serem distribuídos, onde cada pacote m tem um peso w_m e deve ser entregue no ponto p_m. Seja N o número de veículos da frota da empresa, tendo o veículo n uma capacidade de transporte u_n. Suponha que é conhecida uma rota r_n para cada veículo n, sendo esta caracterizada por um conjunto de pacotes que podem ser entregues pelo veículo n e onde c_n é o custo para alocar o veículo a r_n. Como você modelaria o problema de alocar veículos de forma que cada pacote faça parte da rota de pelo menos um veículo alocado e tal que o custo total seja o menor possível?

EX 10.8: No problema de rede de abastecimento é dado um conjunto $S = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ de localizações que deverão receber abastecimento. Dois

tubos mestres, um horizontal a ser instalado na altura y e um vertical a ser colocado na posição x, terão como ponto de conexão a coordenada (x, y). Os pontos de abastecimento de S deverão ser conectados ao tubo mestre horizontal (por um tubo secundário vertical) ou ao tubo mestre vertical (por um tubo secundário horizontal), conforme ilustrado na Figura 10.9.

Nosso problema é determinar o ponto de instalação (x, y) e decidir como que os pontos de abastecimento serão conectados (com o tubo mestre vertical ou horizontal) de forma que a metragem total de tubos secundários seja minimizada.

Formule este problema em programação matemática. É possível colocar o problema em um modelo inteiro misto?

Faça uso do modelo desenvolvido para resolver a instância dada pela Figura 10.9, cujos pontos são: $(x_1, y_1) = (239, 491)$; $(x_2, y_2) = (193, 455)$; $(x_3, y_3) = (318, 478)$; $(x_4, y_4) = (276, 429)$; $(x_5, y_5) = (203, 406)$; $(x_6, y_6) = (149, 379)$; $(x_7, y_7) = (151, 350)$; $(x_8, y_8) = (230, 340)$; $(x_9, y_9) = (325, 347)$; $(x_{10}, y_{10}) = (383, 272)$; $(x_{11}, y_{11}) = (414, 360)$; $(x_{12}, y_{12}) = (440, 442)$; $(x_{13}, y_{13}) = (383, 506)$; e $(x_{14}, y_{14}) = (463, 496)$.

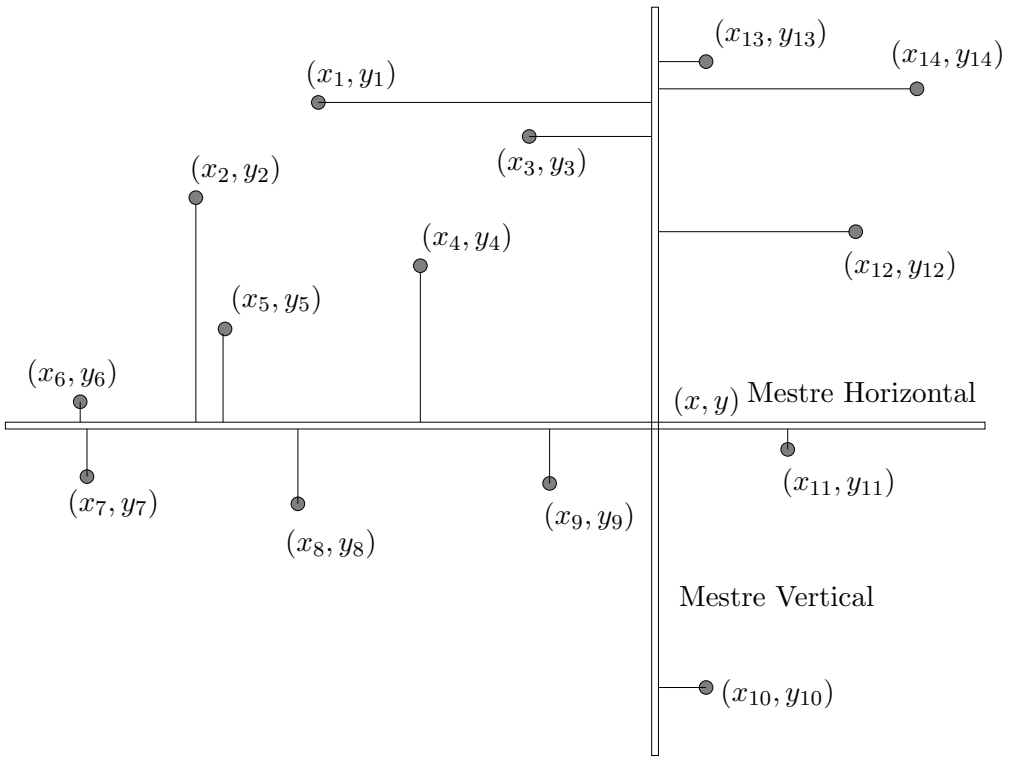


Figura 10.9: Rede de abastecimento.

Capítulo 11

Programação Inteira: Relaxações e Algoritmo *Branch-and-Bound*

Da mesma forma que em problemas de otimização diferenciáveis, é de grande valor conhecermos condições sob as quais uma solução candidata pode ter sua qualidade estimada em relação à solução ótima, mesmo sem conhecermos a solução ótima. Este tipo de estimação é comumente obtido através da solução ótima de problemas mais fáceis, todavia com espaço de soluções mais abrangente do que o espaço do problema em questão. Estes problemas são ditos relaxações, podendo ser obtidos por meio da desconsideração de algumas restrições, da desconsideração das restrições de integralidade, e da simplificação do problema.

Neste capítulo investigaremos questões relacionadas à otimalidade, diferentes tipos de relaxações, e por fim apresentaremos o algoritmo de *branch-and-bound* que pode ser visto como um procedimento de enumeração implícita que faz uso de uma mais relaxações.

O texto a seguir é uma síntese traduzida do livro texto de Wolsey [44].

11.1 Condições de Otimalidade

Inicialmente, considere o problema PI que segue:

$$PI: \qquad z = \text{Max } \{c^T x : x \in X \subseteq \mathbb{Z}^n\}$$

Dado PI e uma solução candidata x^*, como podemos provar que x^* é uma solução ótima? Em outras palavras, procuramos condições de otimalidade que provêm uma condição de parada para algoritmos. Um método que a primeira vista parece "ingênuo", consiste em encontrar um limite inferior $\underline{z} \leqslant z$ e um limite superior $\overline{z} \geqslant z$ tal que se $\underline{z} = \overline{z}$, então z é ótimo. Tipicamente, algoritmos produzem duas seqüências de limites:

- uma seqüência de limites superiores $\overline{z}_1 > \overline{z}_2 > \ldots > \overline{z}_t \geqslant z$
- uma seqüência de limites inferiores $\underline{z}_1 < \underline{z}_2 < \ldots < \underline{z}_k \leqslant z$

Assim dado um $\xi \geqslant 0$, podemos definir um critério de parada como $|\overline{z}_t - \underline{z}_k| \leqslant \xi$. No momento que o algoritmo termina, teremos certeza que a solução candidata z é no máximo ξ unidades inferior à solução ótima. A habilidade de se estimar a qualidade de uma solução candidata é de fundamental importância para otimização.

11.1.1 Limite Primal

Encontrar uma solução factível pode ser uma tarefa fácil ou árdua, dependendo intrinsecamente do problema e da instância particular. Qualquer solução viável x, $x \in X$, induz um limite inferior (limite primal para o caso de maximização) já que $c^T x \leqslant c^T x^* = z^*$. No caso de problema NP-Completos, como o de encontrar um circuito Hamiltoniano em um grafo, a busca de uma solução factível se resulta a resolver o problema enquanto, por outro lado, encontrar uma rota factível do problema do caixeiro viajante é fácil, estando a dificuldade em encontrar a rota mais curta.

11.1.2 Limite Dual

Um método para encontrar limites superiores faz uso de uma relaxação do problema original, ou seja, um problema mais simples cuja solução ótima não é inferior à solução ótima do problema original.

Definição 11.1 *Um problema (RP) $z^R = Max\{f(x) : x \in T \subseteq \mathbb{R}^n\}$ é uma relaxação do problema (PI) $z = Max\{c(x) : x \in X \subseteq \mathbb{R}^n\}$ se:*

i) $X \subseteq T$

ii) $f(x) \geqslant c(x)$ *para todo* $x \in X$.

Proposição 11.1 *Se (RP) é uma relaxação de (PI) então* $z^R \geqslant z$.

11.2 Relaxação Baseada em PL (Relaxação Linear)

Definição 11.2 *Para o problema de programação inteira $Max\{c^T x : x \in P \cap \mathbb{Z}^n\}$ com formulação $P = \{x \in \mathbb{R}^n_+ : Ax \leqslant b\}$ a relaxação em programação linear é $z^{PL} = Max\{c^T x : x \in P\}$.*

Exemplo:

$$\begin{array}{rrrrrr} z = & Max & 4x_1 & -x_2 & & \\ & s.a: & 7x_1 & -2x_2 & \leqslant & 14 \\ & & & x_2 & \leqslant & 3 \\ & & 2x_1 & -2x_2 & \leqslant & 3 \end{array}$$

$$x \in \mathbb{Z}^2_+$$

Limite inferior: observe que $\underline{x} = (2, 1)$ é factível, portanto $z \geqslant 7$.

limite superior: a solução ótima da relaxação linear é $x^* = \left(\frac{20}{7}, 3\right)$ com $z^{LP} = \frac{59}{7}$, portanto concluímos que $z \leqslant 8 \leqslant \frac{59}{7}$, onde $8 = \lfloor \frac{59}{7} \rfloor$.

Proposição 11.2 *Sejam P_1, P_2 duas formulações para o problema inteiro $z = Max\{c^T x : x \in X \cap \mathbb{Z}^n\}$ sendo P_1 mais apertada do que P_2 (ou seja $P_1 \subseteq P_2$). Se $z_i^{PL} = Max\{c^T x : x \in P_i\}$ para $i = 1, 2$, então $z_1^{PL} \leqslant z_2^{PL}$.*

Proposição 11.3

(i) Se a relaxação RP é infactível, então o problema original PI é infactível

(ii) Seja x^* uma solução ótima para RP. Se $x^* \in X \cap \mathbb{Z}^n$ e $f(x^*) = c(x^*)$, então x^* é uma solução ótima para PI.

11.3 Relaxação Combinatória

Sempre que a relaxação é um problema de otimização combinatória, dizemos que ela é uma relaxação combinatória. A seguir exemplificamos a relaxação combinatória no problema do caixeiro viajante e no da mochila.

11.3.1 O Problema do Caixeiro Viajante

Lembrarmos incialmente a formulação em programação matemática do problema:

$$(PCV) \quad Minimize \quad \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} \tag{11.1}$$

$$s.a: \quad \sum_{j=1}^{n} x_{ij} = 1 \qquad i = 1,..,n \tag{11.2}$$

$$\sum_{i=1}^{n} x_{ij} = 1 \qquad j = 1,..,n \tag{11.3}$$

$$\sum_{i\in S}\sum_{j\notin S} x_{ij} \geqslant 1 \qquad \forall S \subset \{1,\ldots,n\}, |S| \geqslant 2 \tag{11.4}$$

$$x_{ij} \in \{0,1\} \qquad \forall (i,j) \in A \tag{11.5}$$

Eliminando a família de restrições (11.4), obtém-se um problema de alocação, conforme o problema de alocação de pessoas a tarefas visto em capítulo anterior. Note que o problema de alocação pode ser eficientemente resolvido fazendo uso de algoritmos para fluxo em redes de custo mínimo.

11.3.2 O Problema da Mochila

O problema da mochila é expresso em programação matemática na forma:

$$\begin{aligned} (PM) \quad Max \quad & \sum_{j=1}^{n} c_j x_j \\ s.a: \quad & \sum_{j=1}^{n} a_j x_j \leqslant b \\ & x_j \in \{0,1\}, j = 1,\ldots,n \end{aligned} \tag{11.6}$$

Uma primeira relaxação pode ser obtida ao tomarmos os valores inteiros imediatamente inferiores aos pesos dos objetos:

$$\begin{array}{lll} (RP_1) & Max & \sum_{j=1}^{n} c_j x_j \\ & s.a: & \sum_{j=1}^{n} \lfloor a_j \rfloor x_j \leqslant b \\ & & x_j \in \{0,1\}, j = 1, \ldots, n \end{array} \tag{11.7}$$

que por sua vez é equivalente ao problema:

$$\begin{array}{lll} (RP_2) & Max & \sum_{j=1}^{n} c_j x_j \\ & s.a: & \sum_{j=1}^{n} \lfloor a_j \rfloor x_j \leqslant \lfloor b \rfloor \\ & & x_j \in \{0,1\}, j = 1, \ldots, n \end{array} \tag{11.8}$$

11.4 Relaxação Lagrangeana

Primeiramente, considere o problema em programação inteira na forma abaixo:

$$(PI) \quad z = \quad Max\ c^T x \tag{11.9}$$
$$s.a:\ Ax \leqslant b \tag{11.10}$$
$$x \in X \subseteq \mathbb{Z}^n \tag{11.11}$$

Como visto acima, uma relaxação pode ser obtida ao eliminarmos o conjunto de restrições $\{Ax \leqslant b\}$. Em vez de eliminar $\{Ax \leqslant b\}$, a relaxação Lagrangeana incorpora estas restrições na função objetivo como segue:

$$(RL) \quad z = \quad Max\ c^T x + u^T(b - Ax) \tag{11.12}$$
$$s.a: \quad x \in X \tag{11.13}$$
$$u \geqslant 0 \tag{11.14}$$

Proposição 11.4 *Seja* $z(u) = Max\ \{c^T x + u^T(b - Ax) : x \in X\}$. *Então* $z(u) \geqslant z$ *para todo* $u \geqslant 0$.

De acordo a Proposição 11.4, a solução ótima do problema RL para $u \geqslant 0$ qualquer induz um limite superior $z(u) \geqslant z^*$, onde z^* é o valor da solução ótima de PI. Não é difícil de se verificar que $z(u) \geqslant c^T x$, para todo $x \in \mathcal{P} = \{x : x \in X \text{ e } Ax \leqslant b\}$: $c^T x \leqslant c^T x + u^T(b - Ax)$ uma vez que $b - Ax \geqslant 0$ para x factível e $u \geqslant 0$; já que as soluções factíveis de PI também fazem parte do espaço de soluções factíveis de RL, verifica-se que $z(u) \geqslant z^*$. Uma vez que $z(u)$ é um limite superior para z^* naturalmente desejamos minimizá-lo, dando origem ao problema Lagrangeano dual:

$$LD: \quad \underset{u \geqslant 0}{Min} \quad z(u) \quad \cong \quad \underset{u \geqslant 0}{Min} \quad \begin{array}{ll} Max & c^T x + u^T(b - Ax) \\ s.a: & x \in X \end{array}$$

Um aspecto relevante do Lagrangeano dual é o fato do problema ser convexo mas não diferenciável. Um algoritmo que pode ser usado para encontrar uma solução aproximada para LD, e até mesmo uma solução ótima quando certas condições são satisfeitas, é o algoritmo subgradiente. Este algoritmo opera de forma semelhante ao algoritmo de descenso. Dada um vetor de multiplicadores de Langrange u_k, o algoritmo encontra um subgradiente d_k que indica a direção na qual os multiplicadores devem ser aumentados/diminuídos. Assim, o iterando seguinte $u_{k+1} = u_k + \alpha_k d_k$ é obtido por meio do passo $\alpha_k \geqslant 0$. Em virtude da natureza não-diferenciável do problema, não é possível obter condições como as de Armijo.

11.5 Algoritmo *Branch-and-Bound*

O algoritmo "branch-and-bound" (B&B) pode ser entendido como uma extensão da estratégia de divisão e conquista para problemas de natureza inteira mista, ou seja, divida um problema P em um conjunto de subproblemas $\{SP_k\}$ de forma que a solução de P possa ser obtida através da solução dos subproblemas, resolva os subproblemas e, no final, obtenha a solução do problema de interesse. De acordo com o algoritmo B&B, as divisões são feitas iterativamente, sempre observando que os subproblemas devem ser mais fáceis de serem resolvidos que o problema original, além de se procurar descartar subproblemas por meio de enumeração implícita—isto equivale a dizer que, de alguma forma, podemos garantir que a solução ótima não é solução de um certo subproblema e, por conseguinte, podemos descartá-lo.

11.5.1 Estratégia de Divisão e Conquista

Considere o problema:

$$z = Max\{c^T x : x \in S\}$$

Como que o problema acima pode ser "quebrado"em subproblemas menores e depois "recombinados"de maneira a se obter uma solução para o problema original? Abaixo enumeramos algumas proposições que servem de regras gerais para que a "quebra" seja feita de uma forma sistemática e com embasamento teórico.

Proposição 11.5 *Seja $S = S_1 \cup \ldots \cup S_K$ uma decomposição de S em K conjuntos menores. Seja também $z^k = Max\{c^T x : x \in S_k\}$ para $k = 1, \ldots, K$. Então, $z = Max_k z^k$.*

Uma maneira típica de se ilustrar a estratégia de divisão e conquista, obedecendo as premissas da Proposição 11.5, é por meio de uma árvore de enumeração.

Exemplo: Para $S \subseteq \{0,1\}^3$, podemos construir a árvore de enumeração da Figura 11.1. Claramente $S = S_0 \cup S_1$, onde $S_0 = \{x \in S : x_1 = 0\}$ e $S_1 = \{x \in S : x_1 = 1\}$. Estendendo este princípio a mais um nível, podemos subdividir cada um dos subproblemas em subproblemas ainda menores, fazendo $S_0 = S_{00} \cup S_{01}$ e $S_1 = S_{10} \cup S_{11}$, onde $S_{i_1 i_2} = \{x \in S_{i_1} : x_2 = i_2\}$.

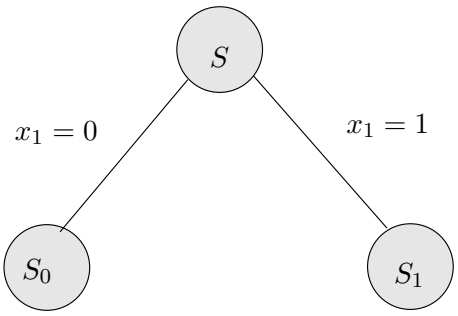


Figura 11.1: Árvore de enumeração

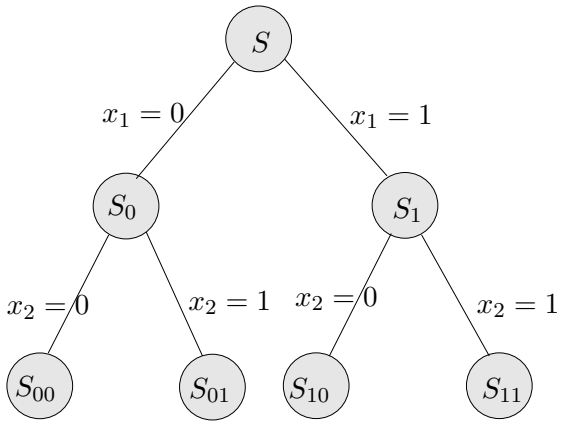


Figura 11.2: Árvore de enumeração

Na Figura 11.3 ilustramos a árvore de enumeração completa. Note que uma folha da árvore $S_{i_1 i_2 i_3}$ é não-vazia se, e somente se, $x = (i_1, i_2, i_3) \in S$. Portanto, as folhas da árvore correspondem precisamente às soluções candidatas que seriam examinadas se fosse conduzida uma enumeração completa.

11.5.2 Enumeração Implícita

Enumeração completa é inviável para problemas práticos—o número de nós cresce exponencialmente. Necessitamos de uma alternativa que vai além da simples divisão do problema em subproblemas. Uma abordagem frequentemente encontrada na prática consiste em se utilizar os limites de $\{z^k\}$ de uma forma inteligente, tanto os limites superiores quanto os inferiores. A proposição 11.6 explicita formalmente como que os limites inferiores e superiores (primais e duais) podem ser empregados de maneira a se realizar enumeração implítica.

Proposição 11.6 *Seja $S = S_1 \cup \ldots \cup S_K$ uma decomposição de S em K subconjuntos, $z^k = Max\{c^T x : x \in S_k\}$ para $k = 1, \ldots, K$, $\overline{z}^k$ um limite superior para z^k e $\underline{z}^k$ um limite inferior para z^k.*

Então:

a) $\overline{z} = Max\{\overline{z}^k : k = 1, \ldots, K\}$ define um limite superior para z

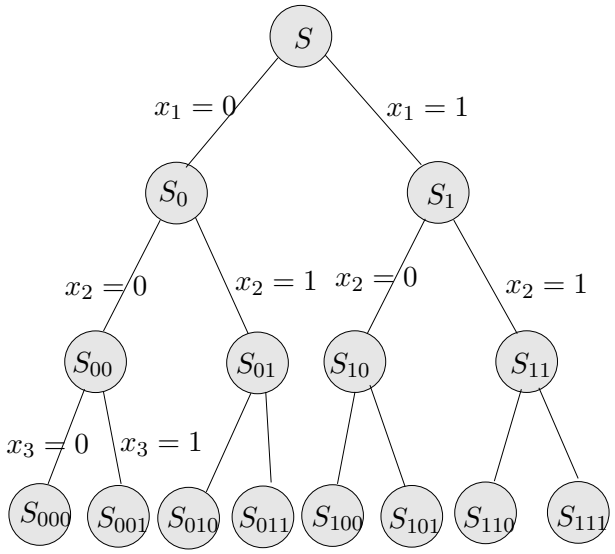


Figura 11.3: Árvore de enumeração completa, explícita

b) $\underline{z} = Max\{\underline{z}^k : k = 1, \ldots, K\}$ *define um limite inferior para* z

Exemplo de corte por otimilidade (maximização)

Vamos ilustrar o funcionamento do algoritmo de *branch-and-bound* em um problema qualquer. Seja S o conjunto inicial contendo todas as soluções do problema, conforme Figura 11.4, sendo o limite inferior $lb = 13$ e o limite superior $ub = 27$. Dividindo o espaço de soluções em dois subconjuntos, $S = S_1 \cup S_2$, e calculando os limites superiores e inferiores para S_1 e S_2, verifica-se que a solução obtida em S_1 é ótima para o problema restrito ao conjunto S_1, já que $l_1 = u_1 = 20$. Portanto, o ramo (S, S_1) foi eliminado por otimalidade. O nó da árvore de enumeração correspondente ao conjunto S_2 de soluções terá de ser examinado uma vez que o limite superior $u_2 = 25 > 20 = l_1$.

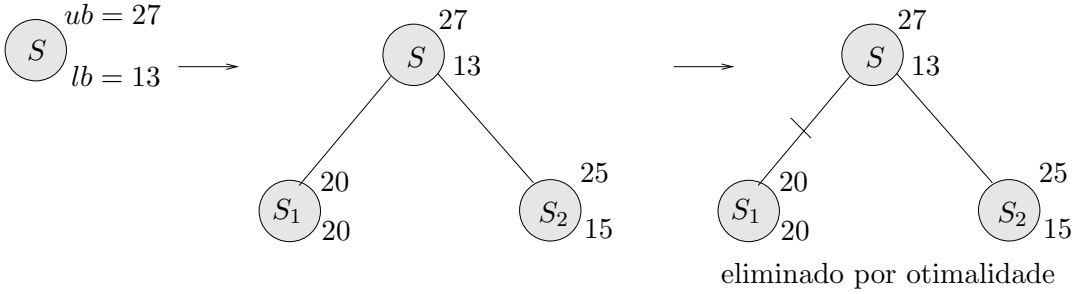


Figura 11.4: Eliminação por otimalidade

Exemplo de corte por limite (maximização)

Considere um problema semelhante ao acima, onde S foi dividido em dois subproblemas tais que $S = S_1 \cup S_2$ conforme Figura 11.5. Limites inferiores e superiores

também foram computados e estão indicados ao lado de cada nó da árvore de enumeração. Note que $l_2 > u_1$, o que significa que a melhor solução possível de ser obtida no espaço de soluções S_1 é inferior a melhor solução primal encontrada até o momento. Portanto, podemos eliminar a busca a partir de S_1 dessa forma eliminando muitas soluções de forma implícita. Este corte é ilustrado na figura.

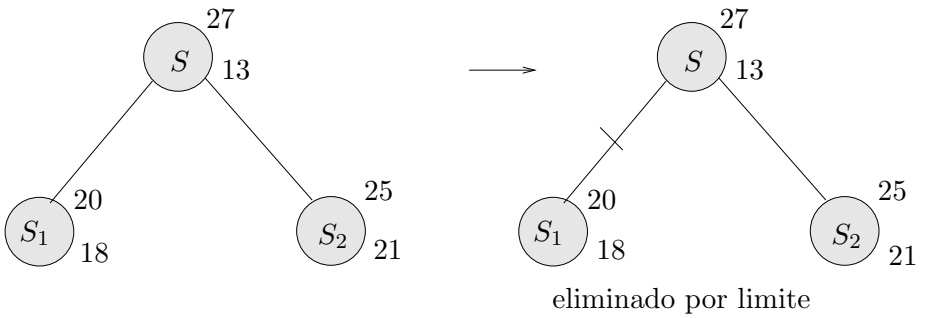


Figura 11.5: Eliminação por limite

Exemplo de situação sem possibilidade de corte

Agora tome como exemplo a árvore de enumeração dada à esquerda da Figura 11.6. Note que o limite superior de S é maior do que o limite superior obtido em S_1 e em S_2, podemos então substituir o limite anterior de 40 para 37, já que $S = S_1 \cup S_2$. O melhor limite inferior obtido até então é 13 que, neste momento, se torna o melhor limite primal para o problema, como ilustrado na árvore à direita. Todavia, ambos os nós S_1 e S_2 devem ser quebrados em subproblemas menores.

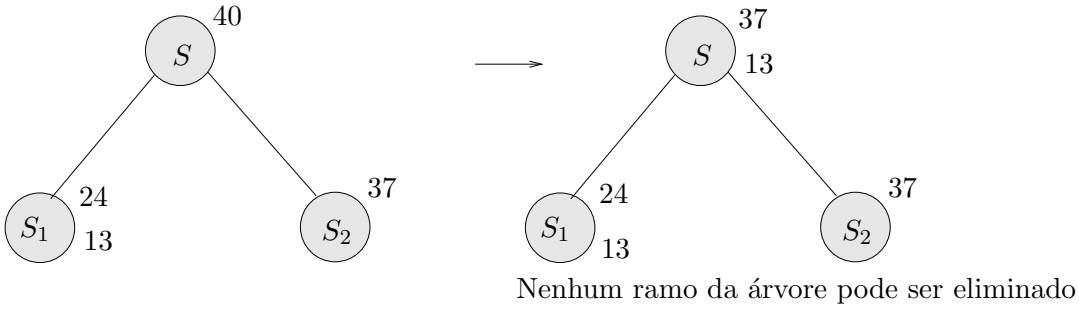


Figura 11.6: Nenhum ramo da árvore pode ser eliminado

Conclusões

Baseado nos exemplos acima, podemos listar três razões para cortar ramos da árvore:

i) Cortando por razão de otimalidade: $z_t = Max\{c^T x : x \in S_t\}$ foi resolvido

ii) Cortando por meio dos limites: $\overline{z}_t < \underline{z}$

iii) Cortando devido à infactibilidade do subproblema: $S_t = \emptyset$

Lembramos aqui que os limites *primais (inferiores)* são obtidos com soluções factíveis, enquanto que os limites *duais (superiores)* são obtidos através de relaxação ou dualidade. É imediata a aplicação da técnica de enumeração com base nas idéias acima apresentadas. No entanto, as seguintes questões são de relevância:

i. Que problema de relaxação ou dual deve ser empregado para obter-se limites superiores? Às vezes podemos tirar vantagem da estrutura do problema na concepção de procedimentos de dualidade como, por exemplo, a relaxação Lagrangeana das restrições de grau 1 do problema do caixeiro viajante.

ii. Como devemos decompor S em $S_1 \cup \ldots \cup S_K$? Não é incomum se fazer valer de uma heurística que explora a natureza particular de um problema.

iii. Em que ordem devemos examinar os subproblemas? Duas estratégias empregadas são a busca em profundidade (*depth-first search*) e a busca pelo melhor ramo (*best-first search*).

11.5.3 Algoritmo *Branch-and-Bound* (B&B)

B&B é um método de enumeração implícita para resolver problemas de programação inteira que utiliza programação linear para calcular os limites duais. Ilustramos o funcionamento do algoritmo através de um exemplo. Consideraremos o problema exemplo abaixo:

$$\begin{array}{rrrrrrr} S: & z = & Max & 4x_1 & -x_2 & & \\ & & s.a: & 7x_1 & -2x_2 & \leqslant & 14 \\ & & & & x_2 & \leqslant & 3 \\ & & & 2x_1 & -2x_2 & \leqslant & 3 \end{array}$$

$$x \in \mathbb{Z}^2_+$$

A aplicação do algoritmo B&B ao problema pode ser ilustrada através dos passos a seguir.

***Bounding*:** Obtemos o primeiro limite superior ao resolvermos a relaxação linear, $R(S)$, obtendo $\overline{z} = \frac{59}{7}$ com $(\overline{x}_1, \overline{x}_2) = (\frac{20}{7}, 3)$. Assumimos que $\underline{z} = -\infty$.

***Branching*:** Uma vez que $\underline{z} < \overline{z}$, devemos quebrar S em subproblemas. Uma idéia é quebrar S no ponto onde uma variável é fracionária, fazendo:

$$\begin{aligned} S_1 &= S \cap \{x : x_j \leqslant \lfloor \overline{x}_j \rfloor\} \\ S_2 &= S \cap \{x : x_j \geqslant \lceil \overline{x}_j \rceil\} \end{aligned}$$

Claramente, $S = S_1 \cup S_2$. Estas operações do algoritmo B&B podem ser observadas na Figura 11.7, onde a quebra foi feita com a variável x_1, fazendo $x_1 \leqslant 2$ ou $x_1 \geqslant 3$. A lista de nós ativos passa a ser $L = \{S_1, S_2\}$.

Escolhendo um nó: A lista de nós ativos $L = \{S_1, S_2\}$ contém dois conjuntos. Arbitrariamente, escolhemos S_1.

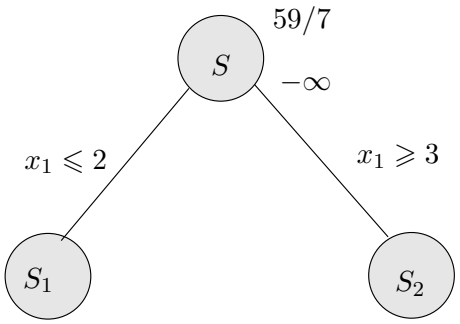


Figura 11.7: Quebra do primeiro nó da árvore de enumeração B&B

***Bounding*:** Resolvemos a relaxação linear $R(S_1)$ associada a S_1, ou seja, resolvemos o problema de programação linear abaixo:

$$\begin{array}{rrrrrrr} S_1: & \overline{z}_1 = & Max & 4x_1 & -x_2 & & \\ & & s.a: & 7x_1 & -2x_2 & \leqslant & 14 \\ & & & & x_2 & \leqslant & 3 \\ & & & 2x_1 & -2x_2 & \leqslant & 3 \\ & & & x_1 & & \leqslant & 2 \end{array}$$

cuja solução ótima é $(x_1^1, x_2^1) = (2, \frac{1}{2})$ que por sua vez induz o limite superior $\overline{z} = \frac{15}{2}$.

***Branching*:** Quebramos S_1 em dois conjuntos: $S_{11} = S_1 \cap \{x : x_2 \leqslant 0\}$ e $S_{12} = S_1 \cap \{x : x_2 \geqslant 1\}$, o que faz com que a lista de nós ativos se torne $L = \{S_2, S_{11}, S_{12}\}$. O resultado desta subdivisão está indicado na Figura 11.8.

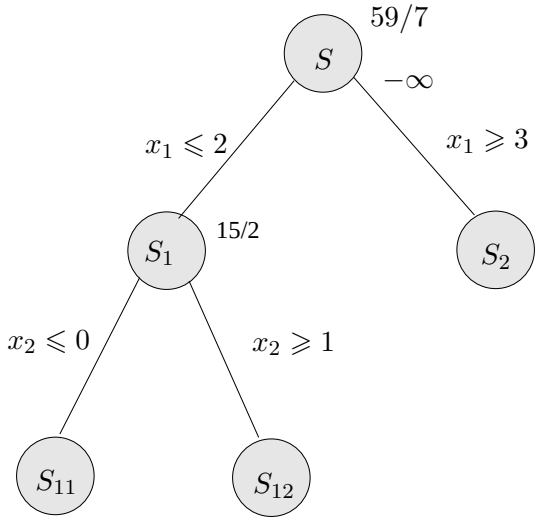


Figura 11.8: Dividindo S_1 em S_{11} e S_{12}

Escolhendo um nó: Arbitrariamente escolhemos o nó identificado por S_2 dentre a lista de nós ativos $L = \{S_{11}, S_{12}, S_2\}$.

***Bounding*:** Resolvemos a relaxação linear $R(S_2)$, correspondendo à relaxação do problema:

$$
\begin{array}{rrrrrrr}
S_2: & z_2 = & Max & 4x_1 & -x_2 & & \\
 & & s.a: & 7x_1 & -2x_2 & \leqslant & 14 \\
 & & & & x_2 & \leqslant & 3 \\
 & & & 2x_1 & -2x_2 & \leqslant & 3 \\
 & & & x_1 & & \geqslant & 3 \\
 & & & & & & \\
 & & & x \in \mathbb{Z}^2_+ & & &
\end{array}
$$

Uma vez que o problema S_2 é infactível, ou seja, $\overline{z}_2 = -\infty$, podemos cortar o nó S_2 em função da inviabilidade.

Escolhendo um nó: A lista de nós ativos passou a ser $L = \{S_{11}, S_{12}\}$. Arbitrariamente escolhemos S_{12}.

***Bounding*:** Resolvemos a relaxação $R(S_{12})$ do problema cujo espaço de soluções factíveis é $S_{12} = S \cap \{x : x_1 \leqslant 2 \text{ e } x_2 \geqslant 1\}$, obtendo a solução $\overline{x}_{12} = (2, 1)$ a qual gera o limite superior $\overline{z}_{12} = 7$. Já que a solução obtida para $R(S_{12})$ é inteira, produzimos o primeiro limite inferior, ou seja, $underline{z}_{12} = 7$ que pode ser propagado para os demais nós da árvore. Portanto o nó S_{12} foi cortado por otimalidade como está indicado na Figura 11.9.

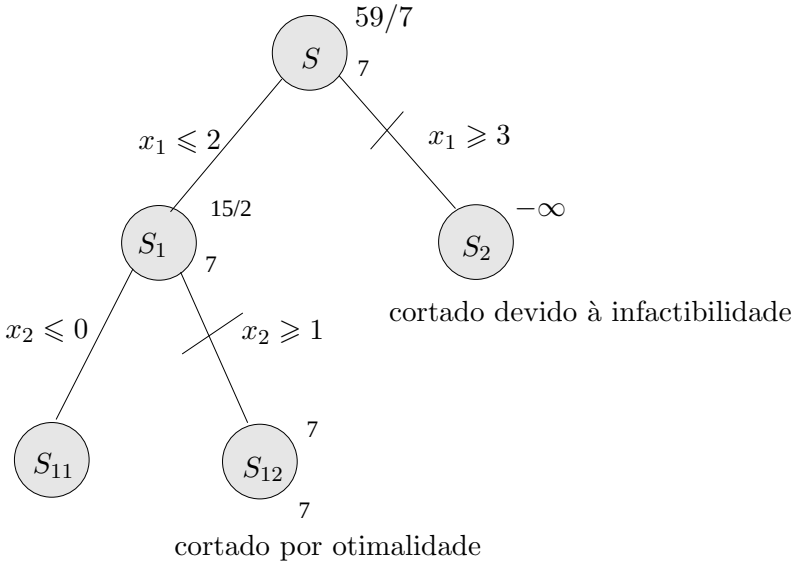


Figura 11.9: Cortando o nó S_{12} por meio da condição de otimalidade

Escolhendo um nó: Apenas S_{11} está ativo.

***Bounding*:** Observe primeiramente que $S_{11} = S \cap \{x : x_1 \leqslant 2, x_2 = 0\}$. Resolvendo a relaxação linear $R(S_{11})$, obtemos a solução $\overline{x}_{11} = (\frac{3}{2}, 0)$ a qual produz o limite superior $\overline{z}_{11} = 6$. Levando em conta que $\overline{z}_{11} = 6 < 7 = \underline{z}$, o nó é cortado.

Escolhendo um nó: Já que a lista está vazia, concluímos que a solução ótima é $x^* = (2, 1)$ com $z^* = 7$.

11.6 Referências

Este capítulo é uma compilação traduzida, incluindo figuras e exemplos, do Capítulo 7 do livro texto de Wolsey [44].

11.7 Exercícios

EX 11.1: Aplique o algoritmo "Branch-and-Bound" ao problema abaixo descrito. Faça uso de um pacote de otimização linear como, por exemplo, `lp_solve` na resolução das relaxações lineares. Desenhe a árvore B&B e escreva os limites (superior e inferior) para cada nó. Tente pré-processar o problema, em outras palavras, tente simplificá-lo.

$$
\begin{array}{lrcl}
Maximize & 77.9x_1 + 76.8x_2 + 89.6x_3 + 97.1y_1 + 31.3y_2 & = & x_0 \\
Sujeito\ a: & & & \\
 & 60.9x_1 + 68.9x_2 + 69.0x_3 - 56.9y_1 + 22.5y_2 & = & 86.5 \\
 & -86.8x_1 + 32.7x_2 + 24.3x_3 + 13.8y_1 - 12.6y_2 & \leqslant & 77.3 \\
 & 10.9x_1 + 3.6x_2 - 40.8x_3 + 43.9y_1 + 7.1y_2 & \leqslant & 82.3 \\
 & x_1, x_2, x_3 \geqslant 0 \text{ e inteiro} & & \\
 & y_1, y_2 \geqslant 0 & &
\end{array}
$$

EX 11.2: (Dual Lagrangeano) Considere o problema da mochila 0-1:

$$
\begin{array}{ll}
max: & 10x_1 + 4x_2 + 14x_3 \\
s.t.: & \\
 & 3x_1 + x_2 + 4x_3 \leqslant 4 \\
 & x_1, x_2, x_3 \in \{0, 1\}
\end{array}
$$

Questões:

- Qual é o valor da solução ótima para a relaxação linear do problema?
- Construa o problema dual Lagrangeano, através da dualização da restrição $3x_1 + x_2 + 4x_3 \leqslant 4$. Qual é o valor ótimo da variável dual e o correspondente valor dual ótimo?
- Por quê os valores ótimos nos dois casos acima são iguais?

EX 11.3: Considere o problema de programação inteira dado por:

$$
\begin{array}{lll}
P: & \text{Maximize} & 17x_1 + 12x_2 \\
 & \text{Sujeito a:} & 10x_1 + 7x_2 \leqslant 40 \\
 & & x_1 + x_2 \leqslant 5 \\
 & & x_1 \leqslant 4 \\
 & & x_2 \leqslant 5 \\
 & & x_1, x_2 \geqslant 0 \\
 & & x_1, x_2 \in \mathbb{Z}
\end{array}
$$

Tarefas:

i. Aplique o algoritmo de *branch-and-bound* e obtenha a solução ótima de P. Ilustre o funcionamento do algoritmo.

ii. Denotando por u_1 e u_2 os multiplicadores para a primeira e segunda restrição, respectivamente, obtenha o subproblema Lagrangeano $L(u_1, u_2)$ dualizando as duas primeiras restrições. Obtenha também o Lagrangeano dual LD. Para $u_1 = 0.5$ e $u_2 = 1$, calcule o limite superior induzido pela relaxação (i.e., resolva o subproblema Lagrangeano). Resolva a relaxação contínua de P (*programação linear*). Compare a solução da relaxação contínua com o limite calculado para $u_1 = 0.5$ e $u_2 = 1$.

Capítulo 12

Algoritmos de Planos de Corte e Desigualdades Fortes

Aqui iniciaremos o estudo de algoritmos *branch-and-cut* (planos de corte) que adicionam desigualdades válidas à relaxação linear até que se obtenha uma solução inteira. Em outras palavras, o algoritmo *branch-and-cut* se assemelha ao algoritmo *branch-and-bound* mas procura aproximar o poliedro inteiro (o fecho convexo das soluções inteiras) de uma forma iterativa. Estudaremos ainda os chamados cortes de Gomory, que podem ser aplicados a qualquer problema linear inteiro (ou misto), e cortes específicos para alguns problemas de interesse.

12.1 Introdução a Planos de Corte

Inicialmente, lembramos o problema inteiro na sua forma geral:

$$IP: \qquad Max\{c^T x : x \in X\}, \quad \text{onde } X = \{x : Ax \leqslant b, x \in \mathbb{Z}^n_+\}$$

Um resultado de significância prática e teória pode ser resumido na propoposição a seguir.

Proposição 12.1 *$conv(X) = \{x : \tilde{A}x \leqslant \tilde{b}, x \geqslant 0\}$ é um poliedro.*

Este resultado nos diz que, pelo menos em teoria, IP pode ser reformulado como um problema de programação linear:

$$LP: \qquad Max\{c^T x : \tilde{A}x \leqslant \tilde{b}, x \geqslant 0\}$$

Note que qualquer ponto extremo de LP é uma solução ótima de IP. Para alguns problemas, tal como o problema de fluxo em redes, conhecemos uma descrição completa de $conv(X)$. Em geral, e particularmente para problemas NP-difíceis, não há esperança de se encontrar uma descrição completa de $conv(X)$, mesmo porque esta pode conter um número exponencial de restrições. Dado um problema NP-difícil, aqui nos preocupamos em encontrar uma aproximação para $conv(X)$ para uma dada instância, podendo esta aproximação ser construída gradualmente à medida que se insere novas restrições válidas e não triviais, preferencialmente desigualdades que toquem no poliedro que descreve $conv(X)$. O conceito fundamental que já utilizamos informalmente é o de desigualdade válida, agora formalizado pela definição abaixo.

Definição 12.1 *Uma desigualdade $\pi^T x \leqslant \pi_0$ é uma desigualdade válida para $X \subseteq \mathbb{R}^n$ se $\pi^T x \leqslant \pi_0$ para todo $x \in X$.*

Surgem imediatamente duas questões relevantes:

a) Quais desigualdades são "boas" e "úteis"?

b) Se conhecemos uma família de desigualdades para um certo problema, como podemos utilizá-las de uma forma eficaz?

Ambas as questões acima serão respondidas nas próximas seções, cujas respostas implicam diretamente na eficácia de uma abordagem por planos de cortes.

12.2 Exemplos de Desigualdades Válidas

No que segue, apresentamos através de exemplos alguns tipos de desigualdades válidas que expressam condições lógicas.

12.2.1 Um conjunto 0-1 puro

Considere o conjunto da mochila 0-1 onde o conjunto de soluções factíveis X (i.e., as soluções que satisfazem a restrição *knapsack*) é dado por:

$$X = \{x \in B^5 : 3x_1 - 4x_2 + 2x_3 - 3x_4 + x_5 \leqslant -2\}$$

Para $x_2 = x_4 = 0$, temos que a desigualdade $3x_1 + 2x_3 + x_5 \leqslant -2$ se torna impossível de ser satisfeita. Portanto, concluímos que uma solução factível deve satisfazer $x_2 + x_4 \geqslant 1$. Se $x_1 = 1$ e $x_2 = 0$, então a desigualdade que resulta $2x_3 - 3x_4 + x_5 \leqslant -5$ não poderá ser satisfeita. Portanto $x_1 \leqslant x_2$ é uma desigualdade válida, podendo esta ser introduzida na formulação de X. A partir das deduções desenvolvidas podemos propor uma formulação revisada para o problema em questão:

$$\begin{aligned} X = \{x \in B^5 : \ & 3x_1 - 4x_2 + 2x_3 - 3x_4 + x_5 \leqslant -2 \\ & x_2 + x_4 \geqslant 1 \\ & x_1 \leqslant x_2\} \end{aligned}$$

12.2.2 Um Conjunto 0-1 Misto

O exemplo de conjunto de soluções mistas (contínuas e discretas) consiste no conjunto X:

$$X = \{(x, y) : x \leqslant 9999y, 0 \leqslant x \leqslant 5, y \in \mathbb{B}\}$$

É fácil verificar que a desigualdade $x \leqslant 5y$ é válida.

12.2.3 Um Conjunto Inteiro Misto

Considere o conjunto

$$X = \{(x, y) : 0 \leqslant x \leqslant 10y, 0 \leqslant x \leqslant 14, y \in \mathbb{Z}_+\}$$

O leitor pode se convencer da validade da desigualdade

$$x \leqslant 14 - 4(2 - y)$$

O espaço de soluções factíveis X juntamente com a desigualdade válida é ilustrado na Figura 12.1.

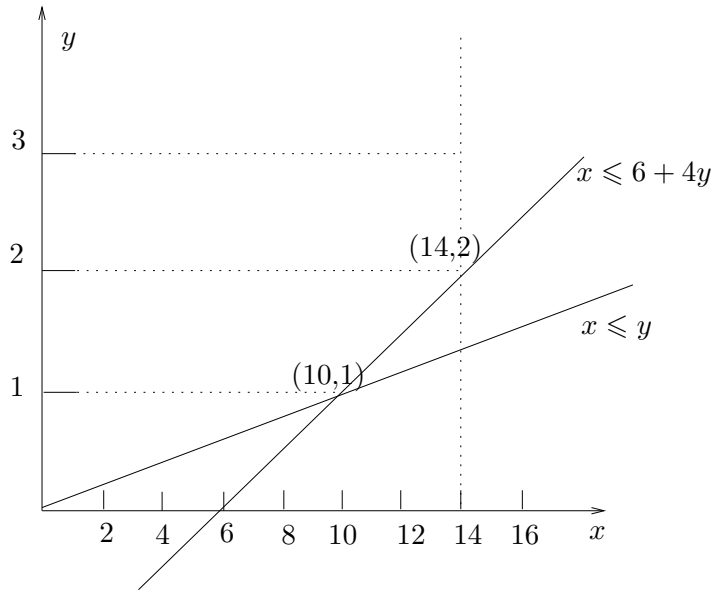


Figura 12.1: Espaço de soluções factíveis e desigualdade válida $x \leqslant 14 - 4(2 - y)$

12.2.4 Conjunto Combinatório

Seja X o conjunto dos vetores de incidência de problemas de emparelhamento ("matchings"), mais precisamente:

$$X = \{x \in \mathbb{Z}_+^{|E|} : \sum_{e \in \delta(i)} x_e \leqslant 1 \qquad \text{para todo } i \in V\}$$

onde $G = (V, E)$ é um grafo não direcionado, e $\delta(i) = \{e \in E : e = (i, j) \text{ para algum } j \in V\}$. Seja ainda $T \subseteq V$ um conjunto qualquer com cardinalidade ímpar. O número de arestas tendo ambos os extremos em T é no máximo $(|T| - 1)/2$, portanto obtemos a desigualdade:

$$\sum_{e \in E(T)} x_e \leqslant \frac{|T| - 1}{2}$$

É propício mencionarmos que $conv(X)$ pode ser obtido através da inserção de todas as desigualdades da forma acima, isto é, $conv(X)$ é precisamente o poliedro dado por:

$$conv(X) = \{x \in \mathbb{R}_+^{|E|} : \begin{array}{ll} \sum_{e\in\delta(i)} x_e \leqslant 1 & \forall i \in V \\ \sum_{e\in E(T)} x_e \leqslant \frac{|T|-1}{2} & \forall T \subseteq V, |T| \text{ ímpar e } |T| \geqslant 3\} \end{array}$$

12.2.5 Arredondamento Inteiro

Considere a região $X = P \cap \mathbb{Z}^4$ e $P = \{x \in \mathbb{R}_+^4 : 13x_1 + 20x_2 + 11x_3 + 6x_4 \geqslant 72\}$. Dividindo a desigualdade por 11, obtemos a seguinte desigualdade válida para P:

$$\frac{13}{11}x_1 + \frac{20}{11}x_2 + \frac{11}{11}x_3 + \frac{6}{11}x_4 \geqslant \frac{72}{11}$$

uma vez que $x \geqslant 0$, podemos arredondar os coeficientes de x para o inteiro mais próximo:

$$\begin{array}{lllllllll} \lceil\frac{13}{11}\rceil x_1 & + & \lceil\frac{20}{11}\rceil x_2 & + & x_3 & + & \lceil\frac{6}{11}\rceil x_4 & \geqslant & \frac{72}{11} \quad \Rightarrow \\ 2x_1 & + & 2x_2 & + & x_3 & + & x_4 & \geqslant & \frac{72}{11} \quad \Rightarrow \\ 2x_1 & + & 2x_2 & + & x_3 & + & x_4 & \geqslant & \lceil\frac{72}{11}\rceil \quad \Rightarrow \\ 2x_1 & + & 2x_2 & + & x_3 & + & x_4 & \geqslant & 7 \end{array}$$

Note que um inteiro que é maior ou igual a $6 + \frac{6}{11}$ deve ser maior ou igual a 7.

12.2.6 Arredondamento Inteiro Misto

Considere o mesmo exemplo anterior mas com a adição de uma variação contínua. Seja $X = P \cap (\mathbb{Z}^4 \times \mathbb{R})$ onde

$$P = \{(y, s) \in \mathbb{R}_+^4 \times \mathbb{R}_+ : 13y_1 + 20y_2 + 11y_3 + 6y_4 + s \geqslant 72\}$$

Novamente dividindo a desigualdade por 11, obtemos

$$\begin{array}{lllllllllll} \frac{13}{11}y_1 & + & \frac{20}{11}y_2 & + & \frac{11}{11}y_3 & + & \frac{6}{11}y_4 & + & \frac{s}{11} & \geqslant & \frac{72}{11} \Rightarrow \\ \frac{13}{11}y_1 & + & \frac{20}{11}y_2 & + & \frac{11}{11}y_3 & + & \frac{6}{11}y_4 & & & \geqslant & \frac{72-s}{11} \end{array}$$

Podemos observar que:

$$\begin{array}{lllllllllll} 2y_1 & + & 2y_2 & + & y_3 & + & y_4 & \geqslant & \lceil\frac{72}{11}\rceil & = & 7 \quad \text{se } s = 0 \\ 2y_1 & + & 2y_2 & + & y_3 & + & y_4 & \geqslant & \lceil\frac{72-6}{11}\rceil & = & 6 \quad \text{se } s = 6 \end{array}$$

Isto sugere que a desigualdade abaixo é válida:

$$2y_1 + 2y_2 + y_3 + y_4 + \alpha s \geqslant 7$$

para algum α. A desigualdade acima pode ser verificada como válida para $\alpha \geqslant \frac{1}{6}$.

12.3 Teoria de Desigualdades Válidas

A presente seção aprofunda os conceitos sobre desigualdades válidas apresentados por meio de exemplos nas seções anteriores.

12.3.1 Desigualdades Válidas para Problemas Lineares

Dado um poliedro $P = \{x : Ax \leqslant b, x \geqslant 0\}$ e uma desigualdade $\pi^T x \leqslant \pi_0$, a primeira questão é se a desigualdade (π, π_0) é válida para P. A proposição abaixo provê condições que garantem a validade de (π, π_0).

Proposição 12.2 *$\pi^T x \leqslant \pi_0$ é válida para $P = \{x : Ax \leqslant b, x \geqslant 0\} \neq \emptyset$ se, e somente se,*

a) existe $u \geqslant 0$ e $v \geqslant 0$ tal que $u^T A - v^T = \pi^T$ e $u^T b \leqslant \pi_0$, ou

b) existe $u \geqslant 0$ tal que $u^T A \geqslant \pi^T$ e $u^T b \leqslant \pi_0$

Prova (b $\Rightarrow$) Se existe $u \geqslant 0$ tal que $u^T A \geqslant \pi^T$ e $u^T b \leqslant \pi_0$, então para qualquer $x \in P$,

$$\begin{aligned} Ax \leqslant b &\Rightarrow u^T Ax \leqslant u^T b \\ &\Rightarrow \pi^T x \leqslant u^T Ax \leqslant u^T b \leqslant \pi_0 \\ &\Rightarrow (\pi, \pi_0) \text{ é uma desigualdade válida.} \end{aligned}$$

■

12.3.2 Desigualdades Válidas para Problemas Inteiros

Proposição 12.3 *A desigualdade $y \leqslant \lfloor b \rfloor$ é válida para $X = \{y \in \mathbb{Z} : y \leqslant b\}$.*

Exemplo: Podemos utilizar a Proposição 12.2 para gerar desigualdades válidas para um poliedro dado pelas restrições abaixo:

$$\begin{aligned} 7x_1 - 2x_2 &\leqslant 14 && (12.1) \\ x_2 &\leqslant 3 && (12.2) \\ 2x_1 - 2x_2 &\leqslant 3 && (12.3) \\ x &\geqslant 0, \quad x \text{ inteiro} && (12.4) \end{aligned}$$

i) Multiplicamos as restrições por um vetor de pesos não negativos $u = (\frac{2}{7}, \frac{37}{63}, 0)$ e obtemos a desigualdade válida:

$$2x_1 + \frac{1}{63}x_2 \leqslant \frac{121}{21}$$

ii) Reduzimos os coeficientes do lado esquerdo ao inteiro mais próximo e obtemos:

$$2x_1 + 0x_2 \leqslant \frac{121}{21}$$

iii) Uma vez que o lado esquerdo só pode assumir valores inteiros, podemos reduzir o valor do lado direito ao inteiro mais próximo, obtendo a desigualdade

$$2x_1 \leqslant \lfloor \frac{121}{21} \rfloor = 5 \;\Rightarrow\; x_1 \leqslant \frac{5}{2} \;\Rightarrow\; x_1 \leqslant 2$$

12.3.3 Procedimento Chvátal-Gomory para Geração de Desigualdades Válidas

O procedimento CG (Chvátal-Gomory) é uma formalização dos passos ilustrados na Seção 12.3.1 que permite *gerar todas as desigualdades válidas de um problema inteiro.* Seja $X = P \cap \mathbb{Z}^n$ o conjunto de soluções onde $P = \{x \in \mathbb{R}^n_+ : Ax \leqslant b\}$ é um poliedro, e $A \in \mathbb{R}^{m \times n}$ é uma matriz com colunas $\{a_1, a_2, \ldots, a_n\}$. Dado $u \in \mathbb{R}^m_+$, o procedimento consiste dos passos que seguem.

Passo 1: A desigualdade $\sum_{j=1}^{n} u^T a_j x_j \leqslant u^T b$ é válida para P uma vez que $u \geqslant 0$ e $\sum_{j=1}^{n} a_j x_j \leqslant b$

Passo 2: A desigualdade $\sum_{j=1}^{n} \lfloor u^T a_j \rfloor x_j \leqslant u^T b$ é válida para P já que $x \geqslant 0$.

Passo 3: A desigualdade $\sum_{j=1}^{n} \lfloor u^T a_j \rfloor x_j \leqslant \lfloor u^T b \rfloor$ é válida para P uma vez que x é inteiro, e portanto $\sum_{j=1}^{n} \lfloor u^T a_j \rfloor x_j$ é inteiro

Importante: um fato surpreendente é que o procedimento acima é capaz de gerar todas as desigualdades válidas para o problema inteiro.

Teorema 12.1 *Toda desigualdade válida para X pode ser obtida por meio da aplicação de um número finito de vezes do procedimento Chvátal-Gomory.*

12.4 O Algoritmo de Planos de Corte

Suponha que $X = P \cap \mathbb{Z}^n$ e que uma família $\mathcal{F}$ de desigualdades válidas $\pi^T x \leqslant \pi_0$, $(\pi, \pi_0) \in \mathcal{F}$, seja conhecida para X. Tipicamente $\mathcal{F}$ contém um número muito grande de elementos (possivelmente exponencial) que não podem ser introduzidas na formulação a priori. Além disso, do ponto de vista prático, não estamos interessados em encontrar uma representação completa do fecho convexo, $conv(X)$, mas sim uma boa aproximação em torno da solução ótima.

Abaixo descreveremos o algoritmo básico de planos de corte para IP, $max\{c^T x; x \in X\}$, que gera cortes "úteis" a partir de $\mathcal{F}$

`Inicialização` Defina $t = 0$ e $P^0 = P$
`Iteração` t Resolva o problema linear $\overline{z}^t = Max\{c^T x : x \in P^t\}$
Seja x^t a solução ótima
Se $x^t \in \mathbb{Z}^n$, pare pois x^t é uma solução ótima para IP
Se $x^t \notin \mathbb{Z}^n$, encontre uma desigualdade $(\pi, \pi_0) \in \mathcal{F}$
tal que $\pi^T x^t > \pi_0$
Se uma desigualdade (π, π_0) foi encontrada,
então faça $P^{t+1} = P^t \cap \{x : \pi^T x \leqslant \pi_0\}$,
incremente t e repita
Caso contrário, pare.

Se o algoritmo termina sem encontrar uma solução inteira, pelo menos $P^t = P \cap \{x : \pi_i^T \leqslant \pi_{i0}, i = 1, 2, \ldots, t\}$ é uma formulação mais "apertada" do que P. Podemos então proceder a partir de P^t com um algoritmo de *branch-and-bound*.

12.5 Algoritmo de Planos de Corte Usando Cortes de Gomory Fracionários

Aqui consideraremos o problema inteiro:

$$Max\{c^T x : Ax = b, x \geqslant 0 \text{ e inteiro}\}$$

O princípio básico é resolver a relaxação linear e encontrar uma base ótima; a partir desta base se escolhe uma variável básica que não seja inteira, e então geramos uma desigualdade Chvátal-Gomory associada a esta variável básica visando "cortá-la", ou seja, eleminá-la do poliedro de relaxação. Supomos que, dada uma base ótima, o problema pode ser reescrito na forma:

$$\begin{array}{ll} Max & \overline{a}_{oo} + \sum\limits_{j \in NB} \overline{a}_{oj} x_j \\ \text{Sujeito a:} & x_{Bu} + \sum\limits_{j \in NB} \overline{a}_{uj} x_j = \overline{a}_{uo} \text{ para } u = 1, \ldots, m \\ & x \geqslant 0 \text{ e inteiro} \end{array}$$

onde 1) $\overline{a}_{oj} \leqslant 0$ para $j \in NB$, 2) $\overline{a}_{uo} \geqslant 0$ para $u = 1, \ldots, m$, e 3) NB é o conjunto de variáveis não básicas, portanto $\{B_u : u = 1, \ldots, m\} \cup NB = \{1, \ldots, n\}$.

Se a solução básica ótima x^* não for inteira, então deve existir uma linha u tal que $\overline{a}_{uo} \notin \mathbb{Z}$. Escolhendo esta linha, o corte de Chvátal-Gomory para a linha u fica:

$$x_{Bu} + \sum_{j \in NB} \lfloor \overline{a}_{uj} \rfloor x_j \leqslant \lfloor \overline{a}_{uo} \rfloor \tag{12.5}$$

Reescrevendo (12.5) de forma a eliminar x_{Bu}, obtemos:

$$x_{Bu} = \overline{a}_{uo} - \sum_{j \in NB} \overline{a}_{uj} x_j \tag{12.6}$$

$$\overline{a}_{uo} - \sum_{j \in NB} \overline{a}_{uj} x_j + \sum_{j \in NB} \lfloor \overline{a}_{uj} \rfloor x_j \leqslant \lfloor \overline{a}_{uo} \rfloor \quad \Rightarrow \tag{12.7}$$

$$\sum_{j \in NB} \left(\overline{a}_{uj} - \lfloor \overline{a}_{uj} \rfloor \right) x_j \geqslant \overline{a}_{uo} - \lfloor \overline{a}_{uo} \rfloor \tag{12.8}$$

De uma forma mais compacta, podemos reescrever o corte (12.8) como:

$$\sum_{j \in NB} f_{uj} x_j \geqslant f_{uo} \tag{12.9}$$

onde $f_{uj} = \overline{a}_{uj} - \lfloor \overline{a}_{uj} \rfloor$ e $f_{uo} = \overline{a}_{uo} - \lfloor \overline{a}_{uo} \rfloor$. Uma vez que $0 \leqslant f_{uj} < 1$ e $0 < f_{uo} < 1$, e $x_j^* = 0$ para toda a variável $j \in NB$ na solução x^*, a desigualdade (12.9) corta a solução corrente x^*.

12.5.1 Exemplo

Considere o problema inteiro a seguir

$$\begin{array}{rrrrrrrl} z = Max & 4x_1 & - & x_2 & & & & \\ s.a.: & 7x_1 & - & 2x_2 & \leqslant & 14 & & \\ & & & x_2 & \leqslant & 3 & & \\ & 2x_1 & - & 2x_2 & \leqslant & 3 & & \\ & x_1, & & x_2 & \geqslant & 0, & \text{inteiros} & \end{array} \tag{12.10}$$

Após adicionarmos variáveis de folga x_3, x_4 e x_5, podemos aplicar o método simplex e obter a solução ótima

$$\begin{array}{rrrrrrrrrrrl} z = Max & \frac{59}{7} & & & - & \frac{4}{7}x_3 & - & \frac{1}{7}x_4 & & & & \\ s.a: & & x_1 & & + & \frac{1}{7}x_3 & + & \frac{2}{7}x_4 & & = & \frac{20}{7} & (= 2.8571) \\ & & & x_2 & & & + & x_4 & & = & 3 & \\ & & & & - & \frac{2}{7}x_3 & + & \frac{10}{7}x_4 & + \; x_5 & = & \frac{23}{7} & (= 3.2857) \\ & x_1, & x_2, & & & x_3, & & x_4, & & \geqslant & 0 & \text{e inteiros} \end{array} \tag{12.11}$$

A solução ótima da relaxação linear é $x^* = (\frac{20}{7}, 3, \frac{27}{7}, 0, 0) \notin \mathbb{Z}^5_+$, portanto usamos a primeira linha de (12.11), na qual a variável básica x_1 é fracionária, para gerar o corte

$$x_1 + \lfloor \tfrac{1}{7} \rfloor x_3 + \lfloor \tfrac{2}{7} \rfloor x_4 \leqslant \lfloor \tfrac{20}{7} \rfloor \;\; \Rightarrow \;\; x_1 \leqslant 2$$

$$\begin{array}{rcl} x_1 + s = 2,\; x_1 = \frac{20}{7} - \frac{1}{7}x_3 - \frac{2}{7}x_4 & \Rightarrow & \frac{20}{7} - \frac{1}{7}x_3 - \frac{2}{7}x_4 + s = 2 \\ & \Rightarrow & s = 2 - \frac{20}{7} + \frac{1}{7}x_3 + \frac{2}{7}x_4 \\ & \Rightarrow & s = -\frac{6}{7} + \frac{1}{7}x_3 + \frac{2}{7}x_4 \\ & & \text{com } s, x_3, x_4 \geqslant 0 \text{ e inteiros.} \end{array}$$

Adicionando a variável s e o corte $s = -\frac{6}{7} + \frac{1}{7}x_3 + \frac{2}{7}x_4$, podemos obter a nova solução ótima:

$$\begin{array}{rrrrrrrrrrrrrl} z = Max & \frac{15}{2} & & & & & - & \frac{1}{2}x_5 & - & 3s & & & \\ s.a: & & x_1 & & & & & & + & s & = & 2 & \\ & & & x_2 & & & - & \frac{1}{2}x_5 & + & s & = & \frac{1}{2} & \\ & & & & x_3 & & - & x_5 & - & 5s & = & 1 & \\ & & & & & x_4 & + & \frac{1}{2}x_5 & + & 6s & = & \frac{5}{2} & \\ & x_1, & x_2, & x_3, & x_4, & & & x_5, & & s & \geqslant & 0 & \text{e inteiros.} \end{array} \tag{12.12}$$

A solução ótima da relaxação linear acima (12.12), $x = (2, \frac{1}{2}, 1, \frac{5}{2}, 0, 0)$, continua fracionária uma vez que x_2 e x_4 são fracionários. A aplicação do corte de Gomory fracionário na segunda linha produz:

$$\begin{array}{rcl} x_2 + \lfloor -\frac{1}{2} \rfloor x_5 + \lfloor 1 \rfloor s \leqslant \lfloor \frac{1}{2} \rfloor & \Rightarrow & x_2 - x_5 + s \leqslant 0 \\ & \Rightarrow & x_2 - x_5 + s + t = 0,\; t \geqslant 0 \\ & \Rightarrow & (\frac{1}{2} + \frac{1}{2}x_5 - s) - x_5 + s + t = 0 \\ & \Rightarrow & t - \frac{1}{2}x_5 = -\frac{1}{2},\; t \geqslant 0 \end{array}$$

Após adicionarmos a variável $t \geqslant 0$ e o corte $t - \frac{1}{2}x_5 = -\frac{1}{2}$ e reotimizarmos, obtemos a tabela simplex:

$$
\begin{array}{llccccccccccl}
z = Max & 7 & & & & & & - & 3s & - & t & & \\
s.a: & & x_1 & & & & & + & s & & & = & 2 \\
& & & x_2 & & & & + & s & - & t & = & 1 \\
& & & & x_3 & & & - & 5s & - & 2t & = & 2 \\
& & & & & x_4 & & + & 6s & + & t & = & 2 \\
& & & & & & x_5 & & & - & t & = & 1 \\
& & x_1, & x_2, & x_3, & x_4, & x_5, & & s, & & t & \geqslant & 0 \text{ e inteiros}
\end{array}
$$

A solução obtida é ótima pois os valores de todas as variáveis são inteiros. A solução ótima é $x^* = (2, 1, 2, 2, 1)$ cujo valor da função objetivo é $z^* = 7$.

12.6 Desigualdades Disjuntivas

Seja $X = X^1 \cup X^2$, onde $X^i \subseteq \mathbb{R}^n_+$, uma disjunção (união) de dois conjuntos X^1 e X^2. Alguns resultados importantes serão enunciados abaixo.

Proposição 12.4 *Se $\sum_{j=1}^n \pi_j^i x_j \leqslant \pi_0^i$ é uma desigualdade válida para X^i, $i = 1, 2$, então a desigualdade*

$$\sum_{j=1}^{n} \pi_j x_j \leqslant \pi_0$$

é válida para X se $\pi_j \leqslant \min\{\pi_j^1, \pi_j^2\}$ para $j = 1, \ldots, n$ e $\pi_0 \geqslant \max\{\pi_0^1, \pi_0^2\}$

Proposição 12.5 *Se $P^i = \{x \in \mathbb{R}^n_+ : A^i x \leqslant b^i\}$ para $i = 1, 2$ são poliedros não-vazios, então (π, π_0) é uma desigualdade válida para $conv(P^1 \cup P^2)$ se e somente se existem $u_1, u_2 \geqslant 0$ tal que:*

$$
\begin{aligned}
\pi^T &\leqslant u_1^T A^1 \\
\pi^T &\leqslant u_2^T A^2 \\
\pi_0 &\geqslant u_1^T b^1 \\
\pi_0 &\geqslant u_2^T b^2
\end{aligned}
$$

12.6.1 Exemplo

Sejam $P^1 = \{x \in \mathbb{R}^2 : -x_1 + x_2 \leqslant 1,\ x_1 + x_2 \leqslant 5\}$ e $P^2 = \{x \in \mathbb{R}^2 : x_2 \leqslant 4,\ -2x_1 + x_2 \leqslant -6,\ x_1 - 3x_2 \leqslant -2\}$ dois poliedros. Fazendo $u_1 = (2, 1)$ e $u_2 = (\frac{5}{2}, \frac{1}{2}, 0)$ e depois aplicando a Proposição 12.5 obtemos:

$$u_1^T A^1 = \begin{bmatrix} 2 & 1 \end{bmatrix} \begin{bmatrix} -1 & 1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} -1 & 3 \end{bmatrix} \qquad u_1^T b^1 = 7$$

$$u_2^T A^2 = \begin{bmatrix} \frac{5}{2} & \frac{1}{2} & 0 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ -2 & 1 \\ 1 & -3 \end{bmatrix} = \begin{bmatrix} -1 & 3 \end{bmatrix} \qquad u_2^T b^2 = 7$$

Isto nos permite obter a desigualdade $-x_1 + 3x_2 \leqslant 7$ válida para $P^1 \cup P^2$. A Figura 12.2 ilustra esta desigualdade.

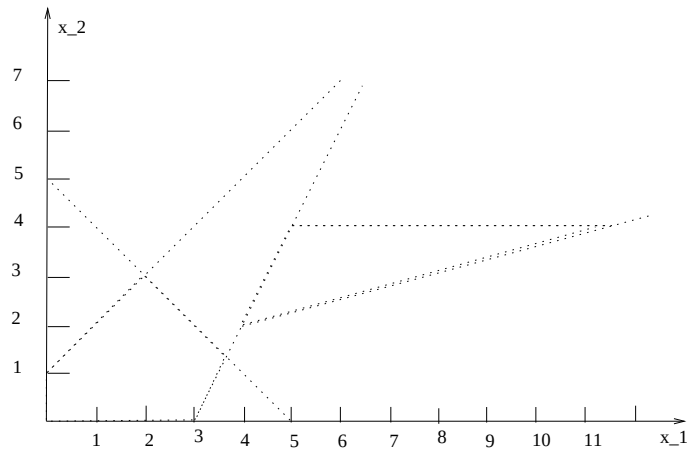


Figura 12.2: Desigualdades disjuntivas

12.6.2 Desigualdades Disjuntivas para Problemas 0-1

Especializando ainda mais, podemos nos restringir a problemas 0-1, onde $X = P \cap \mathbb{Z}^n \subseteq \{0,1\}^n$ e $P = \{x \in \mathbb{R}^n : Ax \leqslant b, 0 \leqslant x \leqslant 1\}$. Seja $P^0 = P \cap \{x \in \mathbb{R}^n : x_j = 0\}$ e $P^1 = P \cap \{x \in \mathbb{R}^n : x_j = 1\}$ para algum $j \in \{1, \ldots, n\}$.

Proposição 12.6 *A desigualdade (π, π_0) é válida para $conv(P^0 \cup P^1)$ se existe $u_i \in \mathbb{R}^m_+$, $v_i \in \mathbb{R}^n_+$, $w_i \in \mathbb{R}^1_+$ para $i = 0, 1$ tal que*

$$\begin{array}{rcl} \pi^T & \leqslant & u_0^T A + v_0 + w_0 e_j \\ \pi^T & \leqslant & u_1^T A + v_1 - w_1 e_j \\ \pi_0 & \geqslant & u_0^T b + \mathbf{1}^T v_0 \\ \pi_0 & \geqslant & u_1^T b + \mathbf{1}^T v_1 - w_1 \end{array}$$

Prova: Aplique a Proposição 12.5 com $P^0 = \{x \in \mathbb{R}^n_+ : Ax \leqslant b, x \leqslant 1, x_j \leqslant 0\}$ e $P^1 = \{x \in \mathbb{R}^n_+ : Ax \leqslant b, x \leqslant 1, -x_j \leqslant -1\}$ ■

Exemplo

Considere a seguinte instância do problema da mochila

$$\begin{array}{lrcrcrcrcl} \text{Max} & 12x_1 & + & 14x_2 & + & 7x_3 & + & 12x_4 & & \\ s.a: & 4x_1 & + & 5x_2 & + & 3x_3 & + & 6x_4 & \leqslant & 8 \end{array}$$

$$x \in \mathbb{B}^4$$

cuja solução linear ótima é $x^* = (1, 0.8, 0, 0)$. Uma vez que $x^*_2 = 0.8$ é fracionário, escolhemos $j = 2$ e definimos P^0 e P^1, e procuramos a desigualdade (π, π_0) que é violada conforme a Proposição 12.6. Para isso, resolvemos um problema de programação linear maximizando $\pi^T x^* - \pi_0$ sobre o poliedro que descreve os coeficientes das desigualdades válidas dadas pela Proposição 12.6.

$$\text{Max} \quad 1.0\pi_1 + 0.8\pi_2 - \pi_0$$

$$s.a: \begin{cases} \pi_1 \leqslant 4u^0 + v_1^0 \\ \pi_1 \leqslant 4u^1 + v_1^1 \\ \pi_2 \leqslant 5u^0 + v_2^0 + w^0 \\ \pi_2 \leqslant 5u^1 + v_2^1 - w^1 \end{cases}$$

$$\begin{cases} \pi_3 \leqslant 3u^0 + v_3^0 \\ \pi_3 \leqslant 3u^1 + v_3^1 \end{cases}$$

$$\begin{cases} \pi_4 \leqslant 6u^0 + v_4^0 \\ \pi_4 \leqslant 6u^1 + v_4^1 \end{cases}$$

$$\begin{cases} \pi_0 \geqslant 8u^0 + v_1^0 + v_2^0 + v_3^0 + v_4^0 \\ \pi_0 \geqslant 8u^1 + v_1^1 + v_2^1 + v_3^1 + v_4^1 - w^1 \end{cases}$$

$$u^0, u^1, v^0, v^1, w^0, w^1 \geqslant 0$$

Objetivando tornar o espaço de soluções factíveis limitado, devemos introduzir um critério de normalização. Duas possibilidades são:

a) $\sum_{j=1}^{n} \pi_j \leqslant 1$

b) $\pi_0 = 1$

Obtemos então a seguinte desigualdade de corte:

$$x_1 + \frac{1}{4}x_2 \leqslant 1.$$

Para P^0, a desigualdade é uma combinação das restrições $x_1 \leqslant 1$ e $x_2 \leqslant 0$ com $v_1^0 = 1$ e $w^0 = \frac{1}{4}$ respectivamente. Para P^1, ela é uma combinação da desigualdade $4x_1 + 5x_2 + 3x_3 + 6x_4 \leqslant 8$ e $-x_2 \leqslant -1$ com $u^1 = \frac{1}{4}$ e $w^1 = 1$, respectivamente.

12.7 Exercícios

EX 12.1: Para os três conjuntos X abaixo, encontre algebricamente uma desigualdade válida cuja adição nos dá $conv(X)$. Verifique graficamente que a adição da desigualdade nos dá $conv(X)$.

i. $X = \{x \in \mathbb{B}^2 : 3x_1 - 4x_2 \leqslant 1\}$

ii. $X = \{(x, y) \in \mathbb{R}_+ \times \mathbb{B} : x \leqslant 20y, x \leqslant 7\}$

iii. $X = \{(x, y) \in \mathbb{R}_+ \times \mathbb{Z}_+ : x \leqslant 6y, x \leqslant 16\}$

EX 12.2: Em cada um dos exemplos abaixo, são dados um conjunto X e um ponto x ou (x, y). Encontre uma desigualdade válida para X que corte (elimine) o ponto.

i. $X = \{(x, y) \in \mathbb{R}_+^2 \times \mathbb{B} : x_1 + x_2 \leqslant 2y, x_1 \leqslant 1; x_2 \leqslant 1\}$ e $(x_1, x_2, y) = (1, 0, 1/2)$

ii. $X = \{(x, y) \in \mathbb{R}_+ \times \mathbb{Z}_+ : x \leqslant 9, x \leqslant 4y\}$ e $(x, y) = (9, 9/4)$

iii. $X = \{(x_1, x_2, y) \in \mathbb{R}^2_+ \times \mathbb{Z}_+ : x_1 + x_2 \leqslant 25, x_1 + x_2 \leqslant 8y\}$ e $(x_1, x_2, y) = (20, 5, 25/8)$

iv. $X = \{x \in \mathbb{Z}^5_+ : 9x_1 + 12x_2 + 8x_3 + 17x_4 + 13x_5 \geqslant 50\}$ e $x = (0, 25/6, 0, 0, 0)$

v. $X = \{x \in \mathbb{Z}^4_+ : 4x_1 + 8x_2 + 7x_3 + 5x_4 \leqslant 33\}$ e $x = (0, 0, 33/7, 0)$

EX 12.3: Prove que a desigualdade $y_2 + y_3 + 2y_4 \leqslant 6$ é válida para $X = \{y \in \mathbb{Z}^4_+ : 4y_1 + 5y_2 + 9y_3 + 12y_4 \leqslant 34\}$

EX 12.4: Considere o problema de programação inteira abaixo:

$$\begin{array}{lll} P: & \text{Maximize} & 17x_1 + 12x_2 \\ & \text{Sujeito a:} & 10x_1 + 7x_2 \leqslant 40 \\ & & x_1 + x_2 \leqslant 5 \\ & & x_1 \leqslant 4 \\ & & x_2 \leqslant 5 \\ & & x_1, x_2 \geqslant 0 \\ & & x_1, x_2 \in \mathbb{Z} \end{array}$$

Aplique o algoritmo de planos de corte ao problema P. Para tanto, tome a relaxação contínua do problema e encontre a solução usando programação linear (Simplex). Se a solução obtida é inteira, então pare. Caso contrário, produza um corte de Gomory, insira na formulação e repita os passos acima. Se você não encontrar a solução inteira ótima após inserir quatro cortes de Gomory, então pare e indique a solução encontrada.

Para cada iteração, mostre o dicionário do Simplex, indique qual variável básica fracionária será utilizada para gerar o corte de Gomory, e mostre o corte gerado.

Capítulo 13

Programação com Restrições Lógicas

Capítulo 14

Programação Dinâmica: Domínio Discreto

Uma técnica poderosa para resolver problemas de decisão sequencial e de controle consiste em quebrá-los em subproblemas menores, mais fáceis de serem resolvidos. Quando quebramos um problema em subproblemas do mesmo tipo, tipicamente podemos produzir algoritmos recursivos, frequentemente empregados para resolver problemas computacionais. Dois paradigmas de concepção de algoritmos por meio de quebra são:

Divisão e Conquista: Este paradigma é amplamente adotado na concepção de algoritmos tipicamente encontrados na Ciência da Computação, podendo ser dividido em três etapas:

- quebre o problema em duas metades;
- resolva cada metade separadamente; e
- combine as metades de forma a obter uma solução completa

O algoritmo de intercalação ou *merge-sort*, por exemplo, ordena uma cadeia de n números em $\Theta(n \log n)$ operações de comparação segundo o princípio de divisão e conquista. O algoritmo "constrói" implicitamente uma árvore de subproblemas, tendo profundidade $\log_2 n$ e largura n. Uma vez que o custo computacional para solução dos subproblemas em cada nível é linear, o algoritmo ordena a cadeia em $\Theta(n \log n)$ operações elementares.

Programação Dinâmica: Este paradigma consiste em quebrar um problema em uma seqüência de decisões que são tomadas em estágios. Em outras palavras, o paradigma segue os passos:

- remova um elemento do problema;
- resolva o problema menor; e
- use a solução do problema menor para adicionar o elemento removido de maneira adequada, produzindo uma solução para o problema maior.

Este tipo de abordagem é comum em problemas de controle ótimo, como será abordado em mais detalhes no capítulo seguinte. Às vezes é possível conceber uma solução analítica fechada para cada estágio o que resulta em um algoritmo eficiente, mas mesmo nos casos onde apenas uma solução iterativa pode ser obtida, a programação dinâmica pode resultar em uma solução algorítmica eficaz. Alguns aspectos importantes sobre DP (Programação Dinâmica) são:

1) problemas cujas instâncias apresentam uma ordem da esquerda para a direita, como cadeias de símbolos e vértices de polígonos, são candidatos a uma solução por DP;

2) sem uma ordem, DP pode resultar em um algoritmo de tempo exponencial; e

3) DP produz uma solução ótima global.

As seções abaixo apresentam o formalismo de programação dinâmica de uma forma informal, através de exemplos. Começamos com o problema de calcular os números de Fibonacci, que ilustra a economia computacional que pode ser obtida por meio da programação dinâmica. O exemplo seguinte mostra como DP pode ser empregada na solução do problema mochila, exemplificando uma maneira de se obter uma solução ótima a partir dos valores ótimos das funções objetivo. O terceiro exemplo trata da identificação da subseqüência de símbolos mais longa, enquanto que o último exemplo apresenta a solução para um problema complexo de edição de cadeias de símbolos. Este último problema e suas generalizações têm aplicações em sistemas operacionais, casamento de padrões e sequenciamento de DNA.

14.1 Um Exemplo de Programação Dinâmica

Programação dinâmica é uma técnica difícil de entender mas cuja solução, uma vez obtida, é de fácil compreensão. Em algoritmos para problemas como o de ordenação de números, corretude é mais fácil de verificar do que eficiência. Este não é o caso de *problemas de otimização*, onde deseja-se provar que um algoritmo sempre produz uma solução ótima. Algoritmos gulosos, heurísticas que executam uma busca local são *eficientes* mas apenas *ocasionalmente* encontram uma solução ótima. Algoritmos baseados em enumeração, por outro lado, avaliam direta- ou indiretamente todas as possíveis soluções e portanto encontram a solução ótima, todavia o custo computacional pode ser proibitivo.

Programação Dinâmica combina estes dois universos (algoritmos gulosos e de enumeração): a técnica sistematicamente considera todas as possíveis decisões e sempre seleciona a ótima, armazenando as consequências de todas as decisões até o presente estágio, e usando estas informações de uma forma sistemática, o trabalho computacional pode ser minimizado.

A melhor maneira de aprender DP é por meio de exemplos.

14.1.1 Calculando Números de Fibonacci

A sequência de Fibonacci é dada por

$$\begin{aligned} F_k &= F_{k-1} + F_{k-2} \\ F_0 &= 0 \\ F_1 &= 1 \end{aligned}$$

A série de Fibonacci foi inicialmente proposta e estudada por um matemático no contexto de reprodução animal. Ela apresenta aplicações em Teoria dos Números e Computação. Uma vez que a fórmula é recursiva, podemos desenvolver um algoritmo recursivo para calcular F_k.

`Algoritmo` $F(k)$

```
if k = 0
   return 0
else
   if k = 1
      return 1
   else
      return F(k − 1) + F(k − 2)
```

Qual é a complexidade do algoritmo? Para calcularmos o número de operações elementares, inicialmente obtemos uma recorrência:

$$\begin{cases} T(n) = 1 & \text{para } n = 0, 1 \\ T(n) = T(n-1) + T(n-2) & \text{para } n \geq 2 \end{cases}$$

Limite inferior para $T(n)$:

$$\begin{aligned} T(n) &\geq T(n-2) + T(n-2) && (14.1) \\ &= 2T(n-2) && (14.2) \\ &\geq 4T(n-4) && (14.3) \\ &\geq 8T(n-6) && (14.4) \\ &\vdots && (14.5) \\ &\geq 2^k T(n-2k) && (14.6) \end{aligned}$$

Note que

$$n - 2k = 1 \Leftrightarrow k = \frac{n-1}{2}$$

portanto, $T(n) \geq 2^{\lfloor \frac{n-1}{2} \rfloor} \Rightarrow T(n) \in \Omega(2^{n/2})$. Conlcuímos então que o algoritmo recursivo é ineficiente, levando um tempo exponencial para calcular F_k. Você pode mostrar que o algoritmo executa em tempo $\Theta(2^n)$? Veja a árvore de recursão dada na Figura 14.1.

Podemos desenvolver um algoritmo mais eficiente, de tempo linear se armazenarmos os valores das instâncias menores e não recalcularmos os seus valores. O algoritmo alternativo é dado abaixo.

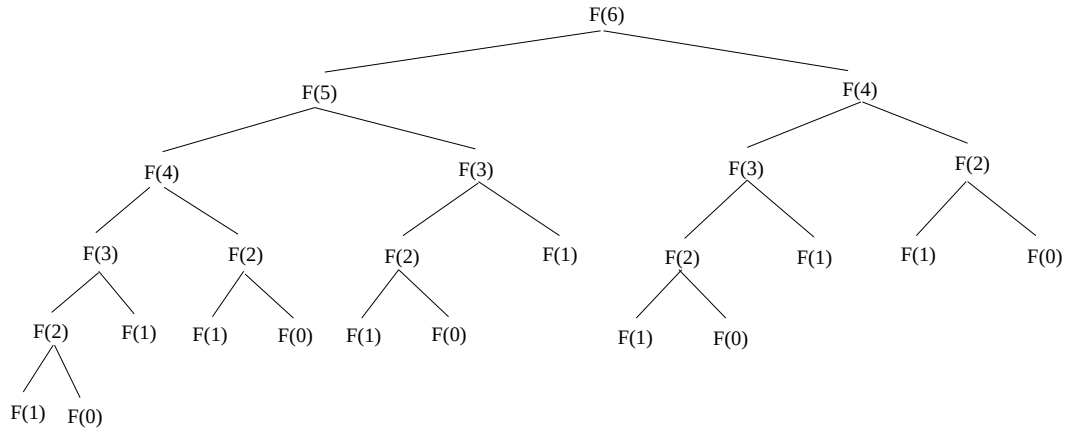


Figura 14.1: Árvore de recursão do algoritmo recursivo para cálculo do número de Fibonacci

`Algoritmo` $F_b(n)$

$F_0 = 0$
$F_1 = 1$
`for` $i \leftarrow 2$ `to` n
    $F_i \leftarrow F_{i-2} + F_{i-1}$
Return F_n

Uma vez que calculamos os números de Fibonacci, começando com os menores e armazenando os resultados, teremos já calculado os valors de F_{i-1} e F_{i-2} quando computarmos o valor de F_i. O algoritmo é executado em tempo $\Theta(n)$.

14.2 Programação Dinâmica para o Problema da Mochila

Inicialmente relembramos o problema da mochila:

$$\begin{aligned} P: z = \quad & Max \quad \sum_{j=1}^{n} c_j x_j \\ & s.\,a: \quad \sum_{j=1}^{n} a_j x_j \leq b \\ & \qquad x_j \in \{0,1\},\ j = 1, \ldots, n \end{aligned}$$

onde os coeficientes $\{a_j\}_{j=1}^{n}$ e b são inteiros positivos.

Imagine que o lado direito da desigualdade assume um valor λ que varia de 0 até b, o que define estágios do problema de programação dinâmica, cujas soluções ótimas são $x_0, \ldots, x_k, \ldots, x_b$. Isto nos leva a definir o problema $P_k(\lambda)$, a sua respectiva solução

ótima $x_k(\lambda)$ e o seu respectivo valor objetivo $f_k(\lambda)$ como segue:

$$\begin{aligned} P_k(\lambda) : f_k(\lambda) = \quad & Max \quad \sum_{j=1}^{k} c_j x_j \\ & s.\,a: \quad \sum_{j=1}^{k} a_j x_j \leq \lambda \\ & \qquad x_j \in \{0,1\},\ j = 1,\ldots,k \end{aligned}$$

Em palavras, $P_k(\lambda)$ é o problema da mochila restrito aos k primeiros itens e uma mochila de capacidade λ. Assim, $z = f_n(b)$ nos dá a solução ótima para P. Nos resta então obter uma recursão que permite calcular $f_k(\lambda)$ em termos dos valores de $f_s(u)$ com $s \leq k$ e $u \leq \lambda$.

O que podemos dizer sobre a solução ótima x^* para o problema $P_k(\lambda)$ com valor $f_k(\lambda)$? Claramente, $x_k^* = 0$ ou $x_k^* = 1$. Considerando cada caso, temos:

Caso 1) Se $x_k^* = 0$, então concluímos que a solução ótima satisfaz $f_k(\lambda) = f_{k-1}(\lambda)$

Caso 2) Se $x_k^* = 1$, então concluímos que a solução ótima satisfaz $f_k(\lambda) = c_k + f_{k-1}(\lambda - a_k)$

Combinando os casos (1) e (2), obtemos a recorrência abaixo:

$$f_k(\lambda) = Max\{f_{k-1}(\lambda),\ c_k + f_{k-1}(\lambda - a_k)\} \tag{14.7}$$

Definindo-se os valores iniciais como $f_0(\lambda) = 0$ para $\lambda \geq 0$, pode-se utilizar a recorrência (14.7) para calcular sucessivamente os valores de $f_1, f_2, \ldots, f_n$ para todos os valores inteiros de $\lambda \in \{0,\ldots,b\}$.

A questão que resta é como encontrar a solução ótima associada ao valor ótimo. Podemos manter um indicador $p_k(\lambda)$ que assume valor 0 se $f_k(\lambda) = f_{k-1}(\lambda)$, e valor 1 caso contrário. A solução pode então ser encontrada por meio dos passos abaixo:

- se $p_n(b) = 0$, então como $f_n(b) = f_{n-1}(b)$, definimos $x_n^* = 0$ e continuamos o processo com o valor $f_{n-1}(b)$;
- se $p_n(b) = 1$, então $f_n(b) = c_n + f_{n-1}(b - a_n)$, definimos $x_n^* = 1$ e repetimos este procedimento para $f_{n-1}(b - a_n)$;
- após n iterações, obteremos a solução ótima.

14.2.1 Complexidade do Algoritmo

Calculando o número de operações necessárias para obtermos $z = f_n(b)$, verificamos que o cálculo de $f_k(\lambda)$ para $\lambda = 0, 1, \ldots, b$ e $k = 1, \ldots, n$ necessita de um número constante de operações. O algoritmo roda em tempo $\Theta(nb)$, sendo portanto pseudo-polinomial. O algoritmo tem tempo de execução polinomial se $b \in O(\log n)$, já que b pode ser representado em notação binária com k bits, ou seja, $b \leq 2^k$.
Exemplo: Considere a instância do problema da mochila que segue abaixo.

$$\begin{array}{rcllllllll} z & = & \text{Maximize} & 10x_1 & + & 7x_2 & + & 25x_3 & + & 24x_4 \\ & & \text{sujeito a:} & 2x_1 & + & x_2 & + & 6x_3 & + & 5x_4 \leq 7 \end{array}$$

Para se aplicar DP, calculamos os valores de $f_0(\lambda)$. A próxima coluna ($k = 1$) é calculada usando a fórmula (14.7). O resultado completo da aplicação do algoritmo de programação dinâmica está ilustrado na Tabela 14.1. Nela são dados os valores da função objetivo de cada subproblema e as decisões relativas à inserção do item do estágio na mochila.

Tabela 14.1: Tabelas DP para o Problema da Mochila

λ	f_0	f_1	f_2	f_3	f_4	p_1	p_2	p_3	p_4
0	0	0	0	0	0	0	0	0	0
1	0	0	7	7	7	0	1	0	0
2	0	10	10	10	10	**1**	**0**	**0**	0
3	0	10	17	17	17	1	1	0	0
4	0	10	17	17	17	1	1	0	0
5	0	10	17	17	24	1	1	0	1
6	0	10	17	25	31	1	1	1	1
7	0	10	17	32	34	1	1	1	**1**

Relembrando que $f_k(\lambda) = Max\{f_{k-1}(\lambda),\ c_k + f_{k-1}(\lambda - a_k)\}$, podemos reconstruir a solução ótima:

$$p_4(7) = 1 \Rightarrow x_4^* = 1$$
$$p_3(7-5) = p_3(2) = 0 \Rightarrow x_3^* = 0$$
$$p_2(2) = 0 \Rightarrow x_2^* = 0$$
$$p_1(2) = 1 \Rightarrow x_1^* = 1$$

A solução ótima é $x^* = (1, 0, 0, 1)$ e $f(x^*) = 34$.

14.3 Elementos de um Algoritmo DP: Sequência Crescente Mais Longa

Aqui ilustramos os elementos básicos de um algoritmo de programação dinâmica. Essencialmente, um algoritmo DP pode ser projetado em três passos:

1) Formule uma resposta como uma relação de recorrência

2) Mostre que o número de valores distintos de cada recorrência é limitada por um polinômio (de preferência de baixa ordem)

3) Especifique uma ordem para avaliar a recorrência de forma que sempre se faça cálculos sobre valores já conhecidos.

Problema: dada uma sequência de n números, e.g. $(9, 5, 2, 8, 7, 3, 1, 6, 4)$, deseja-se encontrar a "subsequência" mais longa cujos números estão em ordem crescente.

No problema em consideração, a sequência mais longa para $s = (9, 5, 2, 8, 7, 3, 1, 6, 4)$ é de comprimento 3, podendo ser a subseqüência $s_1 = (2, 3, 4)$ ou a subseqüência $s_2 = (2, 3, 6)$.

Se estivéssemos procurando uma sequência contígüa, o problema seria facilmente resolvido. **Qual seria o número de soluções candidatas?** O número de soluções candidatas pode ser calculado pela recorrência:

$$\begin{aligned} T(n) &= n + (n-1) + (n-2) + \ldots + 1 && (14.8)\\ &= \frac{n(n-1)}{2} \Rightarrow T(n) \in \Theta(n^2) && (14.9)\end{aligned}$$

Para a situação sob consideração, qual é o número de possíveis soluções candidatas?

$$\begin{aligned} T(n) &= \left(\begin{array}{c} n \\ 1 \end{array}\right) + \left(\begin{array}{c} n \\ 2 \end{array}\right) + \ldots + \left(\begin{array}{c} n \\ n \end{array}\right) && (14.10)\\ &= \sum_{k=1}^{n} \left(\begin{array}{c} n \\ k \end{array}\right) && (14.11)\\ &= (1+1)^n \qquad \text{(pelo Binômio de Newton)} && (14.12)\\ &= 2^n \Rightarrow T(n) \in \Theta(2^n) && (14.13)\end{aligned}$$

Não é difícil construir um algoritmo com tempo de execução $\Theta(n^2)$ para encontrar a subseqüência mais longa de uma cadeia de n símbolos. Tomando como base o número de possíveis soluções para a versão do problema onde a subcadeia não é necessariamente contígüa, $\Theta(2^n)$, nos parece totalmente inviável enumerar todas as possíveis soluções. Veremos a seguir que, apesar do número elevado de soluções candidatas, podemos projetar um algoritmo DP de tempo $\Theta(n^2)$.

14.3.1 Construindo um Algoritmo

Suponha que para a seqüência $s' = (s_1, s_2, \ldots, s_{n-1})$ conhecemos o comprimento l_k, $k \in \{1, \ldots, n-1\}$, da sequência mais longa que termina no elemento s_k de s'. **Como podemos encontrar a sequência mais longa em** $(s_1, \ldots, s_n)$**?** Isto pode ser realizado por meio da recorrência:

$$l_n = Max\{l_k + 1 : s_k \leq s_n,\ k = 1, \ldots, n-1\}$$

para a qual um algoritmo é dado a seguir.

`Algoritmo` $LS(s)$

$l_0 \leftarrow 0$
$s_0 \leftarrow 0$
$p_0 \leftarrow 0$ // `predecessor,` p_i `é o índice do elemento que aparece imediatamente antes de` s_i `na sequência que termina em` s_i.
`for` $k = 1$ `to` n `do`
    $l_k = Max_{j=0,\ldots,k-1}\{l_j + 1 : s_j < s_k\}$
    $p_k = ArgMax_{j=0,\ldots,k-1}\{l_j + 1 : s_j < s_k\}$
`return` $Max\{l_k : k = 1, \ldots, n\}$

O funcionamento do algoritmo durante o cálculo da seqüência mais longa da cadeia $s = (9, 5, 2, 8, 7, 3, 1, 6, 4)$ é ilustrado na Tabela 14.2. O comprimento da cadeia mais longa é, portanto, $3 = Max\{l_i : i = 1, \ldots, n\}$ tendo como solução a subcadeia que termina no elemento 4 ou a subcadeia que termina no elemento 6.

Tabela 14.2: Tabelas DP para o Problema de Subseqüência Mais Longa

	1	2	3	4	5	6	7	8	9
Sequência s_i	0	9	5	2	8	7	3	1	6	4
Comprimento l_i	0	1	1	1	2	2	2	1	3	3
Predecessor p_i	0	0	0	0	2	2	3	0	6	6

Qual é o tempo de execução do nosso algoritmo? Podemos determinar o tempo de execução através da recorrência $T(n) = T(n-1) + n$, o que nos leva a concluir que $T(n) \in \Theta(n^2)$. Utilizando estruturas de dados como dicionários de uma forma inteligente é possível encontrar a "sequência" mais longa em tempo $\Theta(n \log n)$.

14.4 Edição Automática de Cadeias de Símbolos ("Approximate String Matching")

Uma tarefa importante em processamento de texto é a busca por ocorrências de uma palavra no texto. Infelizmente, palavras não são escritas corretamente, surgindo portanto a questão: **Como poderíamos efetuar uma busca pela cadeia mais próxima de um padrão fornecido pelo usuário?** Mais formalmente, seja P uma cadeia padrão e T, uma cadeia que representa o texto. A "distância" entre P e T é o menor número de operações de edição necessárias para transformar T em P, sendo as operações permitidas:

a) Substituição - dois símbolos podem diferir: KAT $\rightarrow$ CAT

b) Inserção - adiciona-se um símbolo de P que não aparece em T: CT $\rightarrow$ CAT

c) Deleção - elimina-se um caracter de T que não aparece em P: CAAT $\rightarrow$ CAT

Exemplo: $P = abcdefghijkl$ pode ser mapeado para $T = bcdeffghixkl$ usando três operações: inserção do caracter a antes de b, deleção de um caracter f, e substituição do caracter x pelo caracter j.

14.4.1 Projeto de um Algoritmo DP

Que informação seria necessária para tomarmos a decisão final? Isto é, o que deve acontecer na última posição de T? Possibilidades

a) o último caracter pode ser "emparelhado" com o último caracter de P, se eles são idênticos

b) de outra forma, o último caracter pode ser substituído

c) as outras operações são de deleção e inserção

Seja $D[i, j]$ a "distância", isto é, o menor número de operações para emparelhamento entre $< P_1, P_2, \ldots, P_i >$ e $< T_1, T_2, \ldots, T_j >$. Então, $D[i, j]$ é o mínimo das três possíveis formas de estender as sub-cadeias.

1) Se $(P_i = T_j)$, então $D[i, j] \leftarrow D[i-1, j-1]$ (*emparelhamento*). Caso contrário, $D[i, j] \leftarrow D[i-1, j-1] + 1$ (*substituição*).

2) $D[i, j] \leftarrow D[i-1, j]+1$, o que significa que há um caracter adicional na cadeia de busca P, portanto pagamos o custo de uma inserção e não avançamos o ponteiro do texto T.

3) $D[i, j] \leftarrow D[i, j-1] + 1$, o que significa que há um caracter extra na cadeia T, portanto pagamos o custo de uma deleção em T e não avançamos o ponteiro de P.

Entretanto, ainda falta especificar as condições iniciais. A inicialização correta dos valores é crítica para que o algoritmo retorne o resultado correto. Por exemplo, o valor $D[0, j]$ corresponde a emparelhar os j primeiros símbolos do texto com nenhum caracter do padrão P.

a) Se desejamos um emparelhamento entre todo o padrão P e todo o texto T, então $D[0, j]$ é o custo de eliminarmos j símbolos de T. Assim, $D[0, j] = j$ para $j = 1, \ldots, |T|$.

b) Se desejamos "emparelhar" P com qualquer subcadeia de T, então o custo de iniciarmos o emparelhamento na posição j do texto deve ser o mesmo de iniciarmos na posição 1. Assim, $D[0, j] = 0$ para $j = 1, \ldots, |T|$.

Em ambos os casos, $D[i, 0] = i$ pois não podemos eliminar símbolos de P sem pagarmos o preço de deleção. Antes de apresentarmos o algoritmo, assumimos que $m = |T|$ e $n = |P|$.

`Algoritmo` $EditDistance(P, T)$

`for` $i = 0$ `to` n `do` $D[i, 0] = i$
`for` $j = 0$ `to` m `do` $D[0, j] = j$

`for`$i = 1$ `to` n `do`
    `for`$j = 1$ `to` m `do`
    $D[i, j] = Min(D[i-1, j-1] + matchcost(P_i, t_i), D[i-1, j] + 1, D[i, j-1] + 1)$
`Fim algoritmo`

Qual é a complexidade do algoritmo? A partir do pseudo-código do algoritmo *EditDistance* fica claro que o tempo de execução é da ordem $\Theta(mn)$.

Exemplo: A tabela abaixo traz o resultado da operação $EditDistance(P, T)$ para $P = abcdefghijkl$ e $T = bcdeftghixkl$.

A Figura 14.2 traz em forma tabular o valores $D[i, j]$, $i = 0, \ldots, n$ e $j = 1, \ldots, m$, calculados pelo algoritmo de programação dinâmica. Abaixo listamos as operações executadas pelo algoritmo, conforme números 1 até 13 indicadas na figura.

1) insert (custo 1)

2) match (custo 0)

3) match (custo 0)

4) match (custo 0)

5) match (custo 0)

6) match (custo 0)

7) delete (custo 1)

8) match (custo 0)

9) match (custo 0)

10) match (custo 0)

11) substitute $x \rightarrow j$ (custo 1)

12) match (custo 0)

13) match (custo 0)

14.5 Exercícios

EX 14.1: Implemente o algoritmo que calcula a distância entre uma cadeia de símbolos padrão $p = (p_1, p_2, \ldots, p_n)$ e um texto $t = (t_1, t_2, \ldots, t_m)$ em qualquer linguagem (e.g., awk, shell script, C/C++, Pascal ou até mesmo Matlab). Produza uma versão onde p pode aparecer em qualquer posição de t e outra onde o "emparelhamento" deve ser perfeito. Ilustre a execução dos algoritmos implementados para uma/duas instâncias.

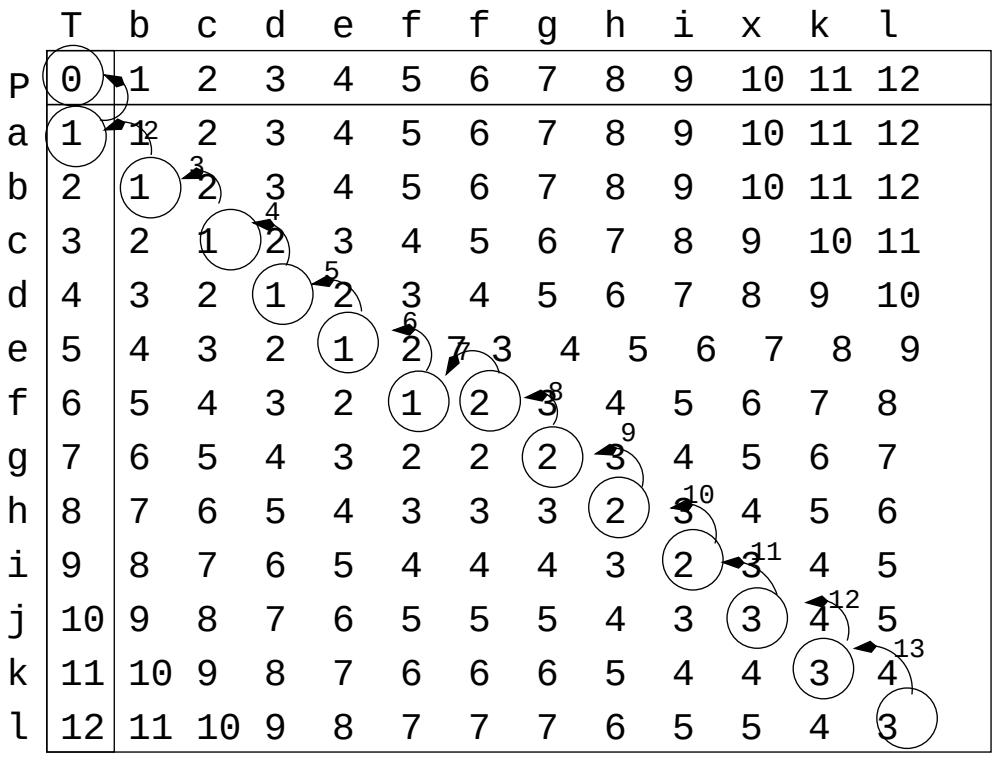


Figura 14.2: Valores $D[i,j]$ calculados pelo algoritmo DP ao calcular a distância de edição mínima entre o padrão $P = abcdefghijkl$ e o texto $T = bcdeffghixkl$

EX 14.2: Considere a seguinte técnica de compressão de texto. Nós é dada uma tabela com m cadeias de símbolos, cada uma de comprimento máximo k. Desejamos representar uma cadeia de dados $s = (s_1, \ldots, s_n)$ usando apenas cadeias da tabela. Por exemplo, se nossa tabela contém $(a, ba, abab, b)$ e a cadeia de dados é $bababbaababa$, então a melhor forma de codificá-la é $(b, abab, ba, abab, a)$, consumindo um total de 5 palavras. Desenvolva um algoritmo de tempo $O(nmk)$ para encontrar a codificação ótima. Assuma que os símbolos elementares (e.g., a e b) aparecem como cadeias na tabela. Ilustre o funcionamento do seu algoritmo em pelo menos dois exemplos.

EX 14.3: Desejamos quebrar uma sequência de N palavras em linhas de comprimento A. Seja $w_1, \ldots, w_N$ as palavras e $L_1, \ldots, L_N$ os respectivos comprimentos destas palavras. Em uma versão simples do problema (desconsiderando hifenação e custos não lineares), as palavras são separadas por espaços brancos de comprimento b, mas espaços brancos podem ser alongadas ou reduzidos se necessário, de forma que uma linha $w_i, w_{i+1}, \ldots, w_{i+k}$ tem comprimento A exato, desde que $\sum_{j=0}^{k} L_{i+j} < A$. O custo associado com a linha é $(k+1)|b' - b|$, onde $b' = (A - L_i - \ldots - L_{i+k})/(k+1)$ é o tamanho médio dos brancos, exceto se a linha é a última linha, sendo neste caso o custo igual a zero se $b' \geq b$. Tarefas:

a) Coloque este problema no formalismo de programação dinâmica. Sugestão: considere os subproblemas de separar as sequências $w_i, \ldots, w_N$, $i = 1, \ldots, N$ de forma ótima.

b) Proponha um algoritmo de programação dinâmica e faça a implementação em uma linguagem de programação (e.g., 'C', Java, Pascal, awk, etc.).

c) Qual é a complexidade (tempo de execução) do algoritmo desenvolvido em termos dos parâmetros dos dados?

d) Teste o algoritmo, indicando os resultados obtidos.

EX 14.4: O problema de caminhos mínimos sob restrição pode ser definido como segue. Dado um grafo direcionado $G = (V, A)$, onde um custo c_{ij} e um comprimento w_{ij} estão associados com cada arco $(i, j) \in A$. Desejamos encontrar o caminho de menor comprimento ligando um vértice s a um vértice t tal que o custo dos arcos ao longo do caminho não supera o montante B. Assuma que todos os valores são inteiros positivos. Tarefas:

a) Formule o problema de forma recursiva.

b) Desenvolva um algoritmo de programação dinâmica. Qual é a complexidade do algoritmo?

c) Implemente o algoritmo em uma linguagem de programação.

d) Ilustre o comportamento do algoritmo em pelo menos dois exemplos. (Sugestão: escolha um grafo com cerca de 15 vértices e rode o algoritmo para diferentes s, t e valores B.)

EX 14.5: São dadas duas seqüências de caracteres do alfabeto $\Sigma = \{A, \ldots, Z\}$, sendo $x = (x_1, \ldots, x_n)$ uma seqüência com n caracteres e $y = (y_1, \ldots, y_m)$ uma seqüência com m caracteres. Desejamos encontrar subseqüências não necessariamente contíguas $x_s = (x_{i_1}, \ldots, x_{i_k})$ de x e $y_s = (y_{j_1}, \ldots, x_{j_k})$ de y que sejam idênticas e de maior comprimento possível, ou seja, queremos maximizar k enquanto que $x_s = y_s$.

Por exemplo, para $x = (A, B, C, B, D, A, B)$ e $y = (B, D, C, A, B, A)$ as subseqüências de maior comprimento são: (B, D, A, B), (B, C, B, A) e (B, C, A, B).

i. Projete um algoritmo eficiente para encontrar o comprimento k da subcadeia mais longa de x e y. Qual é o tempo de execução do seu algoritmo? Quanta memória é necessária para executar o algoritmo? Modifique o seu algoritmo para que ele também produza uma sub-cadeia $x_s = y_s$ de maior comprimento. Qual é o tempo de execução deste algoritmo?

ii. Suponha que você quer obter todas as sub-cadeias mais longas. Como que você resolveria este problema?

iii. Exemplique/ilustre a execução do seu algoritmo para as cadeias x e y dadas acima.

EX 14.6: Prof. Kunz foi contratado pela Empresa XYZ para planejar uma "festa otimizada". A empresa está organizada hierarquicamente com o presidente na raiz da árvore. O departamento de recursos humanos definiu um número real para cada membro da empresa correspondendo ao carácter social de cada indivíduo. Seja $r_m \in \mathbb{R}_+$ o índice social da pessoa m. Para que a festa seja divertida

para todos os convidados, o presidente não quer que os chefes imediatos de cada convidado estejam na festa.

Tarefas:

i. Projete um algoritmo para encontrar um subconjunto $S = \{1, \ldots, N\}$ dos membros da empresa que satifaça a restrição imposta pelo presidente e que maximize a somatória do índices sociais dos convidados. Qual é o tempo de execução do algoritmo? Quanta memória é utilizada pelo algoritmo? Ilustre o funcionamento do algoritmo para a hierarquia dada na Figura 14.3. Os valores r_m são dados em ordem de identificador: 0.9715, 0.1711, 0.1360, 0.7552, 0.3132, 0.4633, 0.3394, 0.0096, 0.3338, 0.2468, 0.9941, 0.9056, 0.3498, 0.2781, 0.0929, 0.2405, 0.9016, 0.3242, 0.5727, 0.1851, 0.7379, 0.1118, 0.3838, 0.8624, 0.8571, 0.8194, 0.2457, 0.8976, 0.6041, 0.1397, 0.1613, 0.8832, 0.9786, 0.6013, 0.3618, 0.2314.

ii. Como que o Prof. Kunz poderia fazer para garantir que o presidente é um dos convidados?

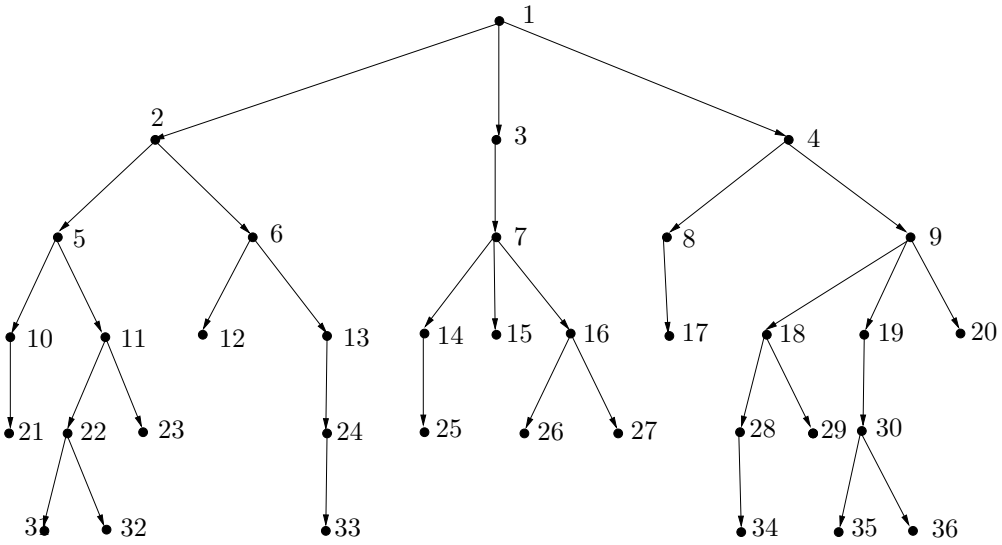


Figura 14.3: Hierarquia da empresa XYZ

EX 14.7: No jogo dos palitos, $n = 21$ palitos são colocados sobre a mesa. Você é o jogador 1. Seja $m_k \in \{1, \ldots, n\}$ o número de palitos na mesa no início da iteração k. Seja $j_k \in \{1, 2\}$ o jogador da vez na iteração k. O valor de j_1 é decidido arbitrariamente (os jogadores decidem quem começa retirando palitos). O seu oponente é o jogador 2. A cada iteração k, o jogador da vez retira l_k palitos tal que $l_k \in \{1, 2, 3\}$ e $l_k \leq m_k$. O jogador que retirar o último palito da mesa perde o jogo.

Formule o jogo dos palitos como um problema de programação dinâmica. Encontre uma política de controle ótima π para o jogador 1, ou seja, encontre uma função $\pi(m_k) = l_k$ que dado o número de palitos na mesa na iteração k retorna o número de palitos a serem removidos.

Capítulo 15

Programação Dinâmica: Domínio Contínuo

Este capítulo se concentra na aplicação de programação dinâmica onde as variáveis de decisão são contínuas, sujeitas a um sistema de equações dinâmicas que descreve o comportamento de um sistema ao longo do tempo, e também susceptíveis a perturbações estocásticas. Problemas de controle, como o problema clássico de regulação e problemas de inventário se enquandram dentro deste formalismo e podem ser resolvidos de forma ótima por meio da programação dinâmica. As próximas seções exemplificam o uso da programação dinâmica na solução do problema de regulação. Conceitos de contrabilidade e observabilidade são também apresentados.

15.1 Introdução

Trataremos aqui de situações onde as decisões são tomadas em estágios. O resultado de cada decisão não pode ser completamente antecipado em virtude de perturbações estocásticas ou eventos não-determinísticos, mas pode ser predito com certo grau de acuracidade antes que uma decisão seja tomada. O objetivo é minimizar um certo custo—uma expressão matemática do que é considerado um resultado indesejável.

Um aspecto crucial é que decisões não podem ser vistas isoladamente, uma vez que se deve balancear o desejo de baixo custo imediato e a possibilidade de altos custos no futuro. A cada estágio, as decisões são ordenadas de acordo com o *custo imediato* e o *custo esperado no futuro*. Programação dinâmica estocástica captura esta relação de compromisso. Tal compromisso surge ainda em outros domínios, como o formalismo de aprendizagem por reforço no qual um agente procura aprender uma política de tomada de decisão que maximiza o ganho amortizado. Neste universo, a questão de balancear o ganho imediato e futuro também desempenha um papel crucial. Há situações onde deve-se pagar um preço alto neste instante se desejamos acumular ganhos futuros.

Existe uma variedade muito grande de problemas que podem ser tratados por DP. Aqui nos concentraremos na sua aplicação no *controle ótimo* de *sistemas dinâmicos*.

O modelo básico do problema apresenta dois elementos essenciais: 1) um sistema dinâmico discreto e 2) uma função custo que é aditiva no tempo. O sistema dinâmico

é da forma:

$$x_k = f_k(x_k, u_k, w_k) \qquad k = 0, 1, \ldots, N-1 \tag{15.1}$$

onde:

- k: índices discretos no tempo;
- x_k: estado do sistema no instante t_k;
- u_k: variável de decisão ou controle para o instante t_k;
- w_k: parâmetro randômico (perturbação); e
- N: horizonte ou número de vezes que o controle é aplicado.

A *função custo* é aditiva, no sentido de que o custo incorrido na iteração k, denotado por $g_k(x_k, u_k, w_k)$, se acumula no tempo. O custo total é dado por:

$$g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, u_k, w_k) \tag{15.2}$$

onde $g_N(x_N)$ é o custo terminal no fim do processo.

No entanto, porque temos elementos estocásticos, o custo é em geral uma variável randômica e não pode ser simplesmente otimizada. Devemos, portanto, otimizar o custo esperado (i.e., a esperança matemática do custo)

$$E\{g_n(x_n) + \sum_{k=0}^{N-1} g_k(x_k, u_k, w_k)\} \tag{15.3}$$

É importante ressaltar que o valor esperado (15.3) é calculado com respeito à distribuição conjunta das variáveis randômicas $w_1, w_2, \ldots, w_{N-1}$. Note também que a otimização é sobre os controles $u_0, u_1, \ldots, u_{N-1}$, que correspondem às variáveis cujos valores podem ser definidos pelo controlador.

15.1.1 Exemplo: Controle de Inventário

Considere o problema de solicitar um certo item de produto a cada um de N períodos de maneira a atender uma demanda estocástica. Aqui, adotamos a notação abaixo:

- x_k: estoque no início do k-ésimo período;
- u_k: quantidade de produto; e
- w_k: demanda durante o período k obedecendo uma certa distribuição de probabilidades.

Assumimos que $w_0, w_1, \ldots, w_{N-1}$ são variáveis randômicas independentes e que o excesso de demanda é suprido tão logo itens adicionais sejam adquiridos. Portanto, o estoque evolui de acordo com a expressão abaixo:

$$x_{k+1} = x_k + u_k - w_k \tag{15.4}$$

O custo incorrido no período k apresenta dois componentes:

a) custo $r(x_k)$ representando a penalidade por estoque positivo (custo de estocagem) ou estoque negativo (demanda não atendida); e

b) custo cu_k correspondente ao custo de adquisição de u_k unidades, onde c é o custo unitário.

Há também um custo terminal $R(x_N)$ para o estoque residual no fim dos N períodos. Portanto, o custo total em N períodos é:

$$E\{R(x_N) + \sum_{k=0}^{N-1}[r(x_k) + cu_k]\} \tag{15.5}$$

Desejamos minimizar (15.5) por meio da escolha de $u_0, u_1, \ldots, u_{N-1}$ sujeito à restrição $u_k \geq 0$ para todo k.

15.1.2 Distinção Entre Otimização "Open-Loop" e "Closed-Loop"

Sob *otimização "open-loop" (em malha aberta)*, escolhemos todos os valores $u_0, u_1, \ldots, u_{N-1}$ de uma só vez, no instante t_0. Por outro lado, sob *otimização "closed-loop" (em malha fechada)*, postergamos a ordem u_k até o último momento, no instante t_k quando conhecemos x_k. Uma vez que não há ônus em postergarmos a decisão u_k, podemos tirar vantagem da informação que se torna disponível.

Otimização em malha fechada é de importância fundamental em programação dinâmica e aqui será objeto de estudo. Portanto, as decisões serão tomadas em estágios, que podem ser postergadas para aprimorar a qualidade das decisões. Na otimização em malha fechada não estamos interessados em encontrar valores numéricos para os pedidos, mas sim *uma estratégia de controle ótima para seleção de u_k a cada período k dado o valor em estoque x_k*. Essa é a diferença entre uma ação e uma estratégia (lei ou regra de controle).

Matematicamente, em otimização por malha fechada, procuramos uma sequência de funções μ_k, $k = 0, 1, \ldots, N-1$, mapeando a quantidade em estoque x_k para uma ordem u_k de forma a minimizar o custo esperado.

$$\mu_k(x_k) = \text{ quantidade a ser solicitada no instante } t_k \text{ se o estoque é } x_k \tag{15.6}$$

Uma sequência $\pi = (\mu_0, \mu_1, \ldots, \mu_{N-1})$ será referenciada como *política* ou *lei de controle*. Para cada π, o correspondente custo para um estoque inicial x_0 é:

$$J_\pi(x_0) = E\{R(x_n) + \sum_{k=0}^{N-1}[r(x_k) + c\mu_k(x_k)]\} \tag{15.7}$$

Um problema típico de programação dinâmica é:

$$Min_\pi \; J_\pi(x_0)$$

Para o problema de inventário que está sendo considerado, uma estratégia ótima é da forma:

$$\mu_k(x_k) = \begin{cases} s_k - x_k & \text{se } x_k < s_k \\ 0 & \text{caso contrário} \end{cases}$$

onde s_k é um limiar determinado a partir dos dados do problema.

15.2 O Problema Básico de Programação Dinâmica

O problema básico de decisão para um número finito de estágios e sob incerteza estocástica é bastante geral, sendo este objeto de estudo no presente capítulo. O problema básico pode ser caracterizado pelos dados abaixo:

a) um sistema dinâmico discreto

$$x_{k+1} = f_k(x_k, u_k, w_k) \qquad k = 0, 1, \ldots, N-1 \tag{15.8}$$

b) o controle u_k deve pertencer a um conjunto não vazio $U_k(x_k) \subseteq C_k$ o qual depende do estado corrente x_k; e

c) a perturbação randômica w_k é caracterizada pela distribuição de probabilidades $P_k(\cdot|x_k, u_k)$ que depende explicitamente de x_k e u_k, mas não depende das perturbações anteriores $w_{k-1}, \ldots, w_0$ (propriedade Markov).

Consideramos as classes de políticas (também conhecidas por leis de controle) que consistem de sequências de funções:

$$\pi = (\mu_0, \mu_1, \ldots, \mu_{N-1}) \tag{15.9}$$

onde μ_k é uma função que mapeia o estado x_k no controle $u_k = \mu_k(x_k)$, tal que $\mu_k(x_k) \in U_k(x_k)$ para todo $x_k \in S_k$. Tais políticas são ditas *admissíveis* pois respeitam as restrições impostas nas variáveis de controle.

Dado um estado x_0 e uma política admissível $\pi = (\mu_0, \ldots, \mu_{N-1})$, o sistema de equações

$$x_{k+1} = f_k(x_k, \mu_k(x_k), w_k) \qquad k = 1, \ldots N-1 \tag{15.10}$$

torna x_k e w_k em variáveis randômicas com distribuições bem definidas. Portanto, dadas funções g_k, o custo esperado pode ser expresso como segue:

$$J_\pi = E_{w_k}\{g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, \mu_k(x_k), w_k)\} \qquad k = 0, 1, \ldots, N-1 \tag{15.11}$$

Note que (15.11) é uma quantidade bem definida. Dado um estado inicial x_0, uma política ótima π^* é aquela que minimiza o custo:

$$J_{\pi^*}(x_0) = Min_{\pi \in \Pi} J_\pi(x_0) \tag{15.12}$$

15.2.1 O Valor da Informação

Com políticas de malha fechada é possível de se induzir um custo reduzido, essencialmente ao tirar-se vantagem da informação adicional (i.e., o valor do estado corrente). A redução no custo em relação às políticas de malha aberta é conhecida como *valor da informação*.

15.3 O Algoritmo de Programação Dinâmica

A técnica de programação dinâmica tem como base o *princípio da otimalidade*. O nome é devido a Bellman, que foi responsável por muitas contribuições neste domínio, um dos precurssores da programação dinâmica.

15.3.1 Princípio da Otimalidade

Seja $\pi^* = (\mu_o^*, \ldots, \mu_{N-1}^*)$ uma política ótima para o problema básico, e assuma que quando adotamos π^*, um dado estado x_i ocorre no instante i com probabilidade positiva. Considere o subproblema em que a partir de x_i no instante i desejamos minimizar o custo restante ("cost-to-go") de i até N. Formalmente o *cost-to-go* é dado por:

$$E\{g_N(x_N) + \sum_{k=i}^{N-1} g_k(x_k, \mu_k^*(x_k), w_k)\} \tag{15.13}$$

Então a política truncada $(\mu_i^*, \mu_{i+1}^*, \ldots, \mu_{N-1}^*)$ é ótima para (15.13). ■

O princípio da otimalidade sugere que uma política ótima pode ser construída de uma forma gradual, primeiro construindo uma política para o último estágio e, depois, estendendo a solução do subproblema para os dois últimos estágios e assim por diante até construirmos uma solução (política) completa, cobrindo todos os estágios. A *programação dinâmica* é baseada nessa idéia.

15.3.2 O Algoritmo de Programação Dinâmica

Considere o problema de inventário apresentado anteriormente. Abaixo desenvolvemos um algoritmo para determinar a política ótima que começa no último estágio e procede para trás.

Período N-1: Independentemente do que aconteceu no passado, devemos minimizar $cu_{N-1} + E\{R(x_N)\}$, que pode ser escrito como:

$$cu_{N-1} + E_{w_{N-1}}\{R(x_{N-1} + u_{N-1} - w_{N-1})\} \tag{15.14}$$

Levando em conta o custo de armazenamento/não suprimento no período $N-1$, o custo ótimo para o último período (mais o custo terminal) é:

$$J_{N-1}(x_{N-1}) = r(x_{N-1}) + Min_{u_{N-1} \geq 0}\left[cu_{N-1} + E\left\{R(x_{N-1} + u_{N-1} - w_{N-1})\right\}\right] \tag{15.15}$$

Naturalmente, J_{N-1} é uma função de x_{N-1}. J_{N-1} pode ser calculada analiticamente ou numericamente, neste último caso uma tabela é utilizada para armazenamento de J_{N-1}. No processo de calcularmos J_{N-1} obtemos a política ótima $\mu_{N-1}^*(x_{N-1})$ para o último período, onde $\mu_{N-1}^*(x_{N-1})$ assume o valor u_{N-1}^* que minimiza (15.15).

Período N-2: Deve ser evidente que devemos minimizar não apenas o custo do período $N-2$ mas sim a soma das parcelas abaixo:

$$\begin{array}{ccccc} \text{O custo esperado} & + & \text{O custo esperado} & = & r(x_{N-2}) + cu_{N-2} \\ \text{do período N-2} & & \text{do período N-1, utilizando-se} & & +E\{J_{N-1}(x_{N-1})\} \\ & & \text{uma política ótima para N-1} & & \end{array} \tag{15.16}$$

Usando-se a equação $x_{N-1} = x_{N-2} + u_{N-2} - w_{N-2}$, o último termo pode ser escrito como $J_{N-1}(x_{N-2} + u_{N-2} - w_{N-2})$. Portanto, se começarmos com x_{N-2}, o custo ótimo para os dois últimos períodos é denotado por $J_{N-2}(x_{N-2})$ e dado pela expressão que segue:

$$J_{N-2}(x_{N-2}) = r(x_{N-2}) + Min_{u_{N-2}\geq 0}\left[cu_{N-2} + E_{w_{N-2}}\left\{J_{N-1}(x_{N-2} + u_{N-2} - w_{N-2})\right\}\right] \tag{15.17}$$

Novamente, $J_{N-2}(x_{N-2})$ é calculado para cada valor x_{N-2}. Simultaneamente, obtemos uma política de controle ótimo $\mu^*_{N-2}(x_{N-2})$.

Observações: Enquanto que o problema de inventário necessita de uma otimização sobre o conjunto de políticas, o algoritmo DP decompõe o problema em uma sequência de minimizações feita sobre um conjunto de controles. Cada uma dessas minimizações é muito mais simples do que o problema original.

15.4 Sistemas Lineares com Custo Quadrático

Esta seção desenvolve um algoritmo de programação dinâmica para resolver o problema de controle que tem como sistema dinâmico equações lineares, uma função de erro quadrática, e um horizonte de tempo finito. Consideramos o caso especial de um sistema linear na forma:

$$x_{k+1} = A_k x_k + B_k u_k + w_k \qquad k = 0, 1, \ldots, N-1 \tag{15.18}$$

tendo como custo a função quadrática dada por:

$$E_{w_k}\left\{x_N{}^T Q_N x_N + \sum_{k=0}^{N-1}\left[x_k{}^T Q_k x_k + u_k{}^T R_k u_k\right]\right\} \qquad k = 0, \ldots, N-1 \tag{15.19}$$

Neste modelo, x_k e u_k são vetores (coluna) de dimensão n e m respectivamente, e as matrizes A_k, B_k, Q_k e R_k são de dimensões apropriadas. Assumimos que as matrizes Q_k são simétricas e positiva semi-definidas, $Q_k = Q_k{}^T$ e $Q_k \geq 0$, e as matrizes R_k são simétricas e positiva definidas, $R_k = R_k{}^T$ e $R_k > 0$. Os controles u_k são irrestritos. As perturbações w_k são variáveis randômicas com distribuição de probabilidades independentes de x_k e u_k. Além disso, as perturbações apresentam média nula $E[w_k] = 0$ e variância finita.

O problema descrito acima é uma formulação popular do problema de regulação, no qual deseja-se manter o estado do sistema próximo da origem.

Duas generalizações do problema acima:

a) Perturbação com média diferente de zero e aproximação de trajetória

$$E\left\{(x_N-\overline{x}_N)^T Q_N(x_N-\overline{x}_N)+\sum_{k=0}^{N-1}\left[(x_k-\overline{x}_k)^T Q_k(x_k-\overline{x}_k)+{u_k}^T R_k u_k\right]\right\}$$

onde $\overline{x}_k$ descreve a trajetória desejada.

b) Problema que surge quando A_k e B_k são matrizes aleatórias com distribuições conhecidas.

15.4.1 Aplicando o Algoritmo de Programação Dinâmica

Aplicando o algoritmo DP, obtemos

$$J_N(x_N)={x_N}^T Q_N x_N \tag{15.20}$$

e

$$J_k(x_k)=Min_{u_k}E\{{x_k}^T Q_k x_k+{u_k}^T R_k u_k+J_{k+1}(A_k x_k+B_k u_k+w_k)\} \tag{15.21}$$

Uma vez que o "cost-to-go" J_k é uma função quadrática, a lei de controle ótima é uma função linear do estado. Escrevendo a equação (15.21) para $k=N-1$, obtemos

$$\begin{aligned}J_{N-1}(x_{N-1})= \quad Min_{u_{N-1}}E\{&{x_{N-1}}^T Q_{N-1}x_{N-1}+{u_{N-1}}^T R_{N-1}u_{N-1}+\\&+(A_{N-1}x_{N-1}+B_{N-1}u_{N-1}+w_{N-1})^T Q_N\\&(A_{N-1}x_{N-1}+B_{N-1}u_{N-1}+w_{N-1})\}\end{aligned} \tag{15.22}$$

Usando o fato de que $E\{w_{N-1}\}=0$, podemos reescrever (15.22) como

$$\begin{aligned}J_{N-1}(x_{N-1})= \quad &{x_{N-1}}^T Q_{N-1}x_{N-1}+Min_{u_{N-1}}\{{u_{N-1}}^T R_{N-1}u_{N-1}+\\&+{u_{N-1}}^T {B_{N-1}}^T Q_N B_{N-1}u_{N-1}+2{x_{N-1}}^T {A_{N-1}}^T Q_N B_{N-1}u_{N-1}\}+\\&+{x_{N-1}}^T {A_{N-1}}^T Q_N A_{N-1}x_{N-1}+E\{{w_{N-1}}^T Q_N w_{N-1}\}\end{aligned} \tag{15.23}$$

Diferenciando (15.23) e forçando o gradiente para zero, produzimos:

$$(R_{N-1}+{B_{N-1}}^T Q_N B_{N-1})u_{N-1}=-{B_{N-1}}^T Q_N A_{N-1}x_{N-1} \tag{15.24}$$

Já que $(R_{N-1}+{B_{N-1}}^T Q_N B_{N-1})$ é positiva definida, esta admite uma inversa e, portanto:

$$u^*_{N-1}=-(R_{N-1}+{B_{N-1}}^T Q_N B_{N-1})^{-1}{B_{N-1}}^T Q_N A_{N-1}x_{N-1} \tag{15.25}$$

Substituindo-se (15.25) em (15.23) obtemos:

$$J_{N-1}(x_{N-1})={x_{N-1}}^T K_{N-1}x_{N-1}+E\{{w_{N-1}}^T Q_N w_{N-1}\} \tag{15.26}$$

$$K_{N-1}={A_{N-1}}^T[Q_N-Q_N B_{N-1}(B_{N-1}{}^T Q_N B_{N-1}+R_{N-1})^{-1}{B_{N-1}}^T Q_N]A_{N-1}+Q_{N-1} \tag{15.27}$$

Não é difícil de se verificar que K_{N-1} é uma matriz *simétrica* e *positiva semi-definida*.

Uma vez que J_{N-1} é uma função quadrática positiva semi-definida (salvo uma constante inconsequente), podemos prosseguir de forma similar e obter a partir da equação (15.21) uma lei de controle ótima para o estágio $N-2$. Procedendo desta forma podemos obter uma lei de controle ótima para cada estágio k. Esta lei toma a forma:

$$\mu_k^*(x_k) = L_k x_k \tag{15.28}$$

onde as matrizes de ganho L_k são dadas pela equação:

$$L_k = -({B_k}^T K_{k+1} B_k + R_k)^{-1} {B_k}^T K_{k+1} A_k \tag{15.29}$$

e as matrizes simétricas e positiva definidas K_k são produzidas pela recorrência abaixo:

$$K_N = Q_N \tag{15.30}$$

$$K_k = {A_k}^T [K_{k+1} - K_{k+1} B_k ({B_k}^T K_{k+1} B_k + R_k)^{-1} {B_k}^T K_{k+1}] A_k + Q_k \tag{15.31}$$

A regra de controle (15.28) é simples e atrativa para implementação em aplicações de engenharia: o estado corrente x_k alimenta a entrada por meio da matriz de ganho L_k.

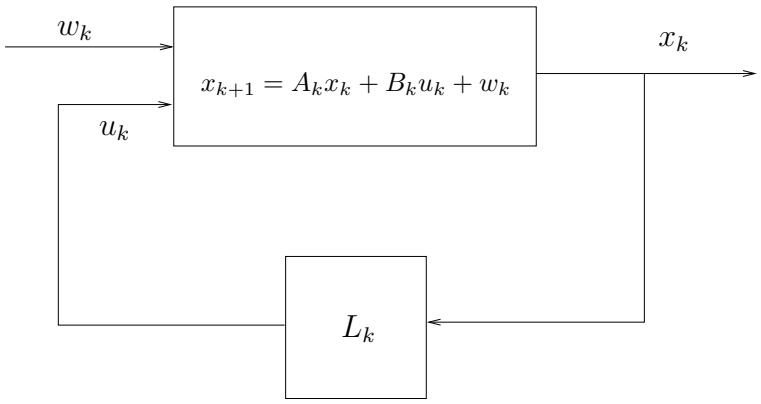


Figura 15.1: Estrutura do sistema linear com controle ótimo de realimentação

15.4.2 A Equação de Riccati e seu Comportamento Assintótico

A equação (15.31) é conhecida como *equação de Riccati*. Ela desempenha um papel importante na teoria de controle. Suas propriedades já foram estudadas amplamente. Uma propriedade importante da equação de Riccati é que a solução K_k converge para uma solução K quando $k \to -\infty$ se A_k, B_k, Q_k e R_k são constantes e iguais a A, B, Q e R, respectivamente. A solução K satisfaz a equação algébrica de Riccati.

$$K = A^T [K - KB(B^T KB + R)^{-1} B^T K] A + Q \tag{15.32}$$

Esta propriedade indica que para o sistema

$$x_{k+1} = Ax_k + Bu_k + w_k \qquad k = 0, 1, \ldots, N-1 \tag{15.33}$$

e um número grande de estágios N, podemos aproximar a lei de controle (15.28) com a lei de controle $(\mu^*, \mu^*, \ldots, \mu^*,)$ onde

$$\mu^*(x) = Lx \tag{15.34}$$

e

$$L = -(B^T K B + R)^{-1} B^T K A \tag{15.35}$$

onde K é uma solução para a equação algébrica de Riccati (15.32).

Definição 15.1 *Um par (A, B), sendo A uma matriz $n \times n$ e B uma matriz $n \times m$, é dito* controlável *se a matriz $n \times nm$*

$$[B, AB, A^2B, \ldots, A^{n-1}B] \tag{15.36}$$

tem posto completo, ou seja, se as linhas são linearmente independentes.

Pode-se demonstrar que se um par (A, B) é controlável, então para qualquer estado inicial x_0 existe uma sequência de vetores $u_0, u_1, \ldots, u_{n-1}$ que força o estado x_n do sistema

$$x_{k+1} = Ax_k + Bu_k \tag{15.37}$$

para a origem no instante n. Isto pode ser demonstrado como segue:

$$\begin{aligned} x_1 &= Ax_0 + Bu_0 \\ x_2 &= Ax_1 + Bu_1 = A^2x_0 + ABu_0 + Bu_1 \\ &\vdots \\ x_n &= A^nx_0 + A^{n-1}Bu_0 + A^{n-2}Bu_1 + \ldots + ABu_{n-2} + Bu_{n-1} \end{aligned} \tag{15.38}$$

ou, equivalentemente,

$$x_n - A^nx_0 = [B, AB, A^2B, \ldots, A^{n-1}B] \begin{bmatrix} u_{n-1} \\ u_{n-2} \\ \vdots \\ u_0 \end{bmatrix} \tag{15.39}$$

Se (A, B) é controlável, a matriz $[B, AB, \ldots, A^{n-1}B]$ tem posto completo e como resultado da equação (15.39) podemos posicionar x_n em qualquer lugar do $\mathbb{R}^n$.

A proposição a seguir mostra que para um sistema estacionário controlável e matrizes constantes Q, R, a solução da equação de Riccati converge para uma matriz simétrica positiva semi-definida a partir de qualquer matriz inicial.

Proposição 15.1 *Para simplificar a notação, revertemos o índice k e, portanto, P_k corresponde à matriz K_{n-k}. Seja A uma matriz $n \times n$, B uma matriz $n \times m$, Q uma matriz $n \times n$ simétrica positiva semi-definida, e R uma matriz $m \times m$ simétrica positiva definida. Considere a equação de Riccati discreta que segue:*

$$P_{k+1} = A^T[P_k - P_kB(B^TP_kB + R)^{-1}B^TP_k]A + Q, \qquad k = 0, 1, \ldots \tag{15.40}$$

sendo que a matriz inicial P_0 é arbitrária, mas simétrica positiva definida. Vamos assumir que o par (A, B) é controlável. Vamos assumir ainda que $Q = C^TC$ para uma matriz C tal que o par (A^T, C^T) é controlável. Então:

a) Existe uma matriz simétrica positiva definida P tal que para qualquer matriz inicial P_0

$$\lim_{k\to\infty} P_k = P$$

Além disso, P é a solução única da equação algébrica de Riccati: $P = A^T[P - PB(B^TPB + R)^{-1}B^TP]A + Q$.

b) O sistema resultante em malha fechada é estável, *i.e., os autovalores da matriz $D = A + BL$ onde $L = -(B^TPB + R)^{-1}B^TPA$ estão estritamente dentro do círculo unitário.* ■

15.5 Exercícios

EX 15.1: Um certo material é submetido a uma sequência de dois fornos, conforme a Figura 15.2. As variáveis são:

- x_0 : temperatura inicial do material;
- x_k $(k = 1, 2)$: temperatura do material na saída do forno k; e
- u_{k-1} $(k = 1, 2)$: temperatura interna no forno k.

Assumimos o modelo dinâmico da forma:

$$x_{k+1} = (1 - \alpha)x_k + \alpha u_k \quad k = 0, 1$$

Onde α é um parâmetro constante e conhecido, sendo que $\alpha \in (0, 1)$. O objetivo é que a temperatura x_2 se aproxime de um valor desejado T, ao mesmo tempo que se minimiza a energia despendida. Esse objetivo pode ser expresso através da função objetivo que segue:

$$r(x_2 - T)^2 + {u_0}^2 + {u_1}^2$$

sendo r uma constante positiva. O sistema é determinístico. Aplique o algoritmo básico de programação dinâmica e obtenha uma regra de controle ótima para operação do sistema de fornos.

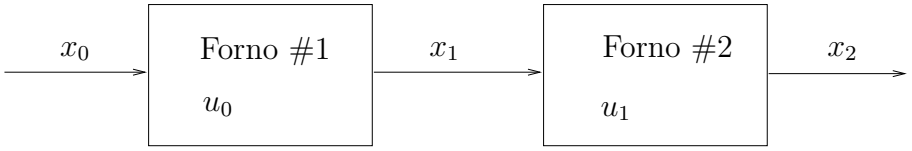


Figura 15.2: Sistema de fornos para aquecimento de materiais

EX 15.2: Verifique que as matrizes K_k, $k = N, N - 1, \ldots, 0$, são simétricas e positiva definidas. Utilize o métdo de indução matemática para desenvolver uma demonstração formal.

EX 15.3: Uma máquina pode estar em um de dois estados: *quebrada* ou *em operação*. Se ela trabalha durante uma semana, então o lucro total obtido é \$100. Se ela quebra durante a semana, então o lucro é nulo. Se a máquina está em operação no início da semana e executamos uma manutenção preventiva, a probabilidade dela falhar durante a semana é 0.4, caso contrário (se não executamos a manutenção) a probabilidade de falha sobe para 0.7. Contudo, a manutenção preventiva custa \$20. Quando a máquina está quebrada no início da semana, ela pode ser reparada a um custo de \$40, podendo neste caso falhar durante a semana com uma probabilidade 0.4, ou substituir a máquina quebrada por uma nova a um custo de \$150, sendo a probabilidade de falha nula neste caso. Encontre uma política de reparo, substituição e manutenção preventiva que maximize o custo total durante quatro semanas. Assuma que na primeira semana a máquina está em operação.

EX 15.4: No jogo dos palitos, $n = 21$ palitos são colocados sobre a mesa. Você é o jogador 1. Seja $m_k \in \{1, \ldots, n\}$ o número de palitos na mesa no início da iteração k. Seja $j_k \in \{1, 2\}$ o jogador da vez na iteração k. O valor de j_1 é decidido arbitrariamente (os jogadores decidem quem começa retirando palitos). O seu oponente é o jogador 2. A cada iteração k, o jogador da vez retira l_k palitos tal que $l_k \in \{1, 2, 3\}$ e $l_k \leq m_k$. O jogador que retirar o último palito da mesa perde o jogo.

Formule o jogo dos palitos como um problema de programação dinâmica. Encontre uma política de controle ótima π para o jogador 1, ou seja, encontre uma função $\pi(m_k) = l_k$ que dado o número de palitos na mesa na iteração k retorna o número de palitos a serem removidos.

EX 15.5: No jogo par-ímpar você deverá rolar dois dados. Se a soma dos valores dos dados for ímpar você ganha R\$ 100, caso contrário você perde R\$ 110. Há dois dados, A e B, que podem ser escolhidos a cada vez. Você rola um dos dados, observa o resultado e depois rola novamente qualquer um dos dados.

Os dados são distintos, apresentando as seguintes probabilidades:

$$\begin{cases} P(A=1)=0.1 & P(A=2)=0.1 & P(A=3)=0.3 \\ P(A=4)=0.3 & P(A=5)=0.1 & P(A=6)=0.1 \end{cases}$$

$$\begin{cases} P(B=1)=0.23 & P(B=2)=0.1 & P(B=3)=0.23 \\ P(B=4)=0.1 & P(B=5)=0.24 & P(B=6)=0.1 \end{cases}$$

Encontre uma política de controle ótima para este jogo que, dado o valor s_k dos dados na iteração k, decide qual dado deve ser rolado com o intuito de maximizar os ganhos. Observe que temos apenas três iterações: $s_0 = 0$ é o valor no início do jogo; s_1 é igual ao resultado do primeiro dado; e s_2 é igual a soma dos valores dos dois dados.

EX 15.6: Para o pêndulo invertido ilustrado na Figura 15.3, a dinâmica do movimento pode ser aproximada em torno do estado $(y, \dot{y}, \theta, \dot{\theta}) = (0, 0, 0, 0)$ pelas equações diferenciais abaixo :

$$\begin{array}{rcl} (M+m)\ddot{y} + ml\ddot{\theta} & = & u \\ 2l\ddot{\theta} - 2g\theta + \ddot{y} & = & 0 \end{array} \tag{15.41}$$

onde $M = 10Kg$, $m = 0.5Kg$, $g = 9.8m/s^2$, e $l = 50cm$. Realizar as seguintes tarefas:

i. Fazendo $x_1 = y, x_2 = \dot{y}, x_3 = \theta, x_4 = \dot{\theta}$ e $x = (x_1, x_2, x_3, x_4)$, obter um sistema de equações diferenciais

$$\dot{x} = Ax + Bu \tag{15.42}$$

equivalente ao sistema de equações (15.41).

ii. Inicialmente, vamos desenvolver o modelo discretizado de (15.42). A partir da definição de derivada, temos que:

$$\dot{x}(t) = \lim_{T \to 0} \frac{x(t+T) - x(t)}{T}$$

Para uma constante de amostragem T, a equação acima pode ser aproximada fazendo:

$$\begin{aligned} \dot{x}(t)T = x(t+T) - x(t) \quad &\Rightarrow \quad x(t+T) = \dot{x}(t)T + x(t) \\ &\Rightarrow \quad x(t+T) = [Ax(t) + Bu(t)]T + x(t) \\ &\Rightarrow \quad x(t+T) = (I + TA)x(t) + TBu(t) \end{aligned}$$

Para $t = 0, T, 2T, \ldots$, e estado inicial $x(0)$, podemos expressar a aproximação discretizada da seguinte forma:

$$x((k+1)T) = (I + TA)x(kT) + TBu(kT), \quad k = 0, 1, \ldots \tag{15.43}$$

Obtenha a aproximação discretizada (15.43) correspondente ao modelo contínuo (15.41). Utilize uma constante de tempo $T = 0.1s$. Para matrizes $Q = I$ e $R = 0.1I$, horizonte de tempo finito $N = 6$, e condição inicial $x(0) = [\ 0.2 \quad 0 \quad 0.5 \quad 0\ ]^T$, aplique a técnica LQR de tempo finito para encontrar uma política de controle ótima. Obtenha a trajetória $(x(0), x(T), \ldots, x(NT))$ induzida pela lei de controle ótima π^*. Calcule o valor de $J_0^*(x(0))$ produzido por π^*.

iii. Repita os passos do item anterior, mas desta vez utilize a lei de controle π_∞ obtida se assumirmos que o horizonte de tempo é infinito. Compare o custo induzido pelas políticas de controle π^* e π_∞.

EX 15.7: No jogo par-ímpar você deverá rolar dois dados. Se a soma dos valores dos dados for ímpar você ganha R$ 100, caso contrário você perde R$ 110. Há dois dados, A e B, que podem ser escolhidos a cada vez. Você rola um dos dados, observa o resultado e depois rola novamente qualquer um dos dados. Os dados são distintos, apresentando as seguintes probabilidades:

$$\begin{cases} P(A=1) = 0.1 & P(A=2) = 0.1 & P(A=3) = 0.3 \\ P(A=4) = 0.3 & P(A=5) = 0.1 & P(A=6) = 0.1 \end{cases}$$

$$\begin{cases} P(B=1) = 0.23 & P(B=2) = 0.1 & P(B=3) = 0.23 \\ P(B=4) = 0.1 & P(B=5) = 0.24 & P(B=6) = 0.1 \end{cases}$$

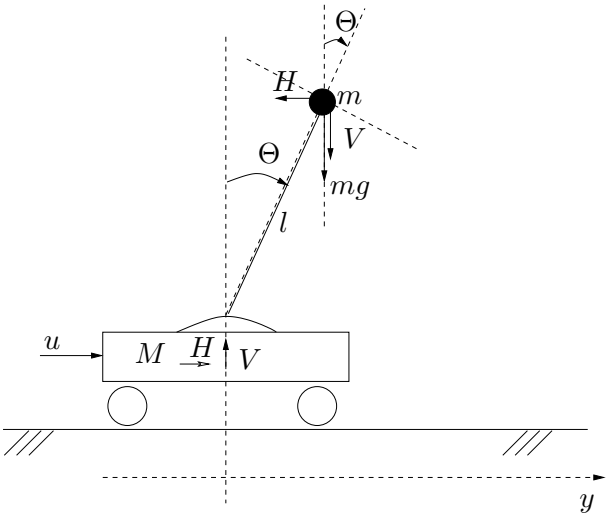


Figura 15.3: Pêndulo invertido

Encontre uma política de controle ótima para este jogo que, dado o valor s_k dos dados na iteração k, decide qual dado deve ser rolado com o intuito de maximizar os ganhos. Observe que temos apenas três iterações: $s_0 = 0$ é o valor no início do jogo; s_1 é igual ao resultado do primeiro dado; e s_2 é igual a soma dos valores dos dois dados.

Capítulo 16

Programação Não-Linear Restrita: Fundamentos e Condições de Otimalidade

Os primeiros capítulos do livro se concentraram na minimização de funções de múltiplas variáveis, onde foram apresentadas condições necessárias e suficientes para que um ponto definisse um ótimo local, para problemas em geral e com estrutura particular, como os problemas convexos. Algoritmos de descenso e de Newton foram desenvolvidos para resolver tais problemas e também encontrar a solução de sistemas de equações não lineares. Tanto no caso de minimização de funções quanto na programação dinâmica no domínio contínuo, não se considerou a possível presença de funções que caracterizam a região factível. O presente capítulo dá início ao estudo dos problemas de minimização de funções não-lineares sob restrições. Serão apresentados conceitos fundamentais, exemplos e condições de otimalidade local de uma forma gradual.

16.1 Teoria da Otimização Não-Linear sob Restrições

Aqui nos concentraremos no problema de minimizar uma função sujeita a restrições. A formulação geral do problema é dada abaixo:

$$\begin{array}{lll} P: \text{ Minimize} & f(x) & \\ \text{Sujeito a :} & & \\ & c_i(x) = 0 & i \in \mathcal{E} \\ & c_i(x) \geq 0 & i \in \mathcal{I} \\ & x \in \mathbb{R}^n & \end{array} \tag{16.1}$$

onde as funções f e c_i, $i \in \mathcal{I} \cup \mathcal{E}$, são suaves e de valor real, $\mathcal{E}$ é o conjunto dos índices das restrições de igualdade e $\mathcal{I}$ é o conjunto dos índices das desigualdades.

O conjunto de soluções factíveis Ω é definido por:

$$\Omega = \{x \in \mathbb{R}^n : c_i(x) = 0,\ i \in \mathcal{E} \text{ e } c_i(x) \geq 0,\ i \in \mathcal{I}\}. \tag{16.2}$$

Assim, podemos reescrever o problema P de uma forma mais compacta:

$$\begin{array}{lll} P: & \text{Minimize} & f(x) \\ & \text{Sujeito a}: & x \in \Omega \end{array} \tag{16.3}$$

Nas próximas seções desenvolveremos uma caracterização matemática das soluções para o problema (16.3).

16.1.1 Solução Local × Solução Global

Já verificamos que uma solução global é difícil de ser encontrada, mesmo na ausência de restrições. Esta situação pode ser melhorada com a introdução de restrições: as restrições podem reduzir o número de mínimos locais. Entretanto, restrições podem tornar o problema muito mais difícil. Como um exemplo, considere o problema abaixo:

$$\begin{array}{lll} P_1: & \text{Minimize} & \|x\|_2{}^2 \\ & \text{Sujeito a}: & \\ & & \|x\|_2{}^2 \geq 1 \end{array} \tag{16.4}$$

Sem a restrição, P_1 é um problema quadrático convexo com um minimizador único $x = 0$. Quando a restrição é introduzida, qualquer vetor x com $\|x\|_2 = 1$ define uma solução. Existe um número infinito de soluções (e portanto um número infinito de mínimos locais) quando $n \geq 2$.

O exemplo a seguir mostra que a introdução de uma restrição produz um grande número de mínimos locais que não formam um conjunto conexo:

$$\begin{array}{lll} P_2: & \text{Minimize} & (x_2 + 100)^2 + 0.01x_1^2 \\ & \text{Sujeito a}: & \\ & & x_2 - \cos x_1 \geq 0 \end{array} \tag{16.5}$$

Sem a restrição, P_2 tem uma solução ótima no ponto $(0, -100)$. Com a restrição, temos ótimos locais próximos dos pontos:

$$(x_1, x_2) = (\pi k, -1) \quad \text{para } k = 0, \pm 1, \pm 2, \cdots \tag{16.6}$$

Dizemos que um vetor x^* é um *ótimo local* do problema (16.3) se $x^* \in \Omega$ e se existe uma vizinhança N em torno de x^*, $N = \{x \in \mathbb{R}^n : \|x - x^*\| \leq \varepsilon\}$ para $\varepsilon > 0$, tal que $f(x) \geq f(x^*)$ para todo $x \in N \cap \Omega$.

16.1.2 Suavidade das Funções

A suavidade da função objetivo e das restrições é uma característica importante na caracterização das soluções. Ela garante que a função objetivo e as restrições se comportam de uma forma previsível, pelo menos localmente, dessa forma permitindo que algoritmos tomem decisões corretas com respeito às direções de busca. Tipicamente, fronteiras não suaves podem ser descritas por uma coleção de restrições suaves. Considere a região da Figura 16.1.

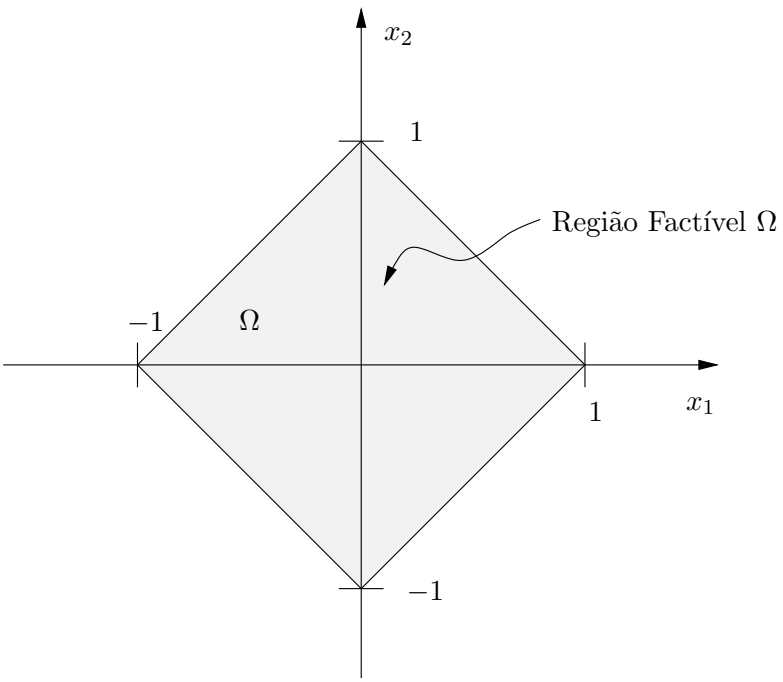


Figura 16.1: Fronteiras não suaves

A região factível Ω pode ser descrita por uma única restrição não-suave:

$$\|x\|_1 = |x_1| + |x_2| \leq 1 \tag{16.7}$$

Ω também pode ser descrita pelas seguintes restrições suaves:

$$\begin{array}{rl} x_1 + x_2 & \leq 1 \\ x_1 - x_2 & \leq 1 \\ -x_1 + x_2 & \leq 1 \\ -x_1 - x_2 & \leq 1 \end{array} \tag{16.8}$$

Problemas sem restrições e não-suaves podem às vezes ser reformulados como problemas suaves com restrições. Considere o problema abaixo:

$$P_3 : f(x) = Max\{x^2, x\} \tag{16.9}$$

Podemos obter uma formulação suave para P_3 por meio da adição da variável t:

$$\begin{array}{lll} P_3 : & \text{Minimize} & t \\ & \text{Sujeito a :} & \\ & & t \geq x \\ & & t \geq x^2 \end{array} \tag{16.10}$$

16.2 Exemplos

Para qualquer ponto factível x, a desigualdade $i \in \mathcal{I}$ é dita *ativa* se $c_i(x) = 0$ e *inativa* se $c_i(x) > 0$.

16.2.1 Uma Restrição de Igualdade

Considere o problema abaixo:

$$\begin{array}{lll} P_4: & \text{Minimize} & x_1 + x_2 \\ & \text{Sujeito a:} & \\ & & x_1^2 + x_2^2 - 2 = 0 \quad (c_1(x) = 0) \end{array} \tag{16.11}$$

O espaço de soluções factí veis é o cí rculo de raio $\sqrt{2}$, com o centro na origem, conforme a Figura 16.2.

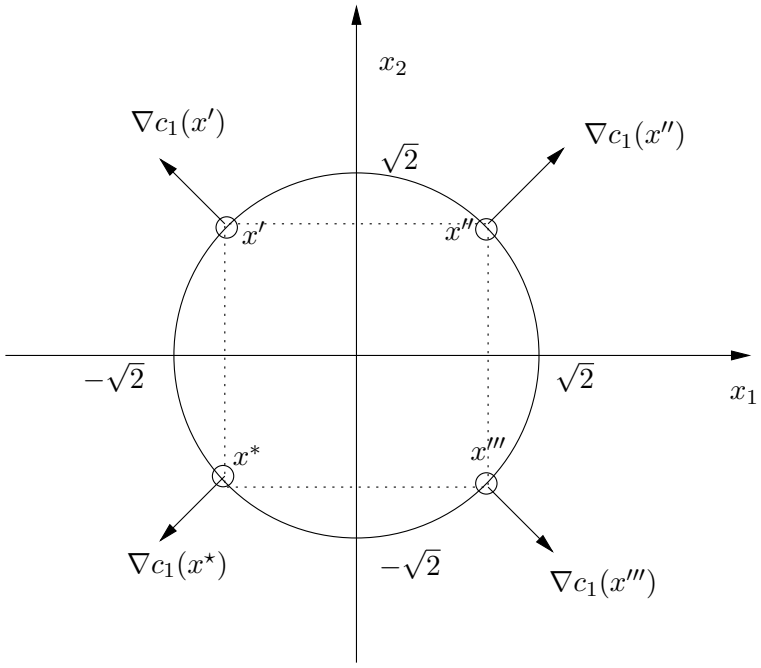


Figura 16.2: Espaço de soluções

A solução ótima é obviamente $x^* = (-1, -1)$. A partir de qualquer outro ponto do círculo, podemos mover a solução dentro do conjunto de soluções factíveis e, ao mesmo tempo, reduzir o valor da função objetivo. Não é difícil de se verificar que no ponto $x = x^*$,

$$\nabla f(x) = \lambda_1 \nabla c_1(x) \tag{16.12}$$

para algum λ_1. Para o caso em questão, $\lambda_1 = -\frac{1}{2}$, conforme segue:

$$\begin{array}{rcl} \nabla f(x^*) & = & [1, 1]^T \\ \nabla c_1(x^*) & = & [2x_1, 2x_2]^T \quad = \quad [-2, -2]^T \\ \lambda_1 & = & -\frac{1}{2} \end{array}$$

Podemos obter a equação (16.12) a partir da série de Taylor. Para manter factibilidade em torno do ponto x, $c_1(x + d) = 0$, deduzimos que:

$$c_1(x + d) \approx c_1(x) + \nabla c_1(x)^T d = \nabla c_1(x)^T d \tag{16.13}$$

Portanto, a direção mantém factibilidade com respeito a c_1, quando d satisfaz:

$$\nabla c_1(x)^T d = 0 \tag{16.14}$$

Similarmente, a direção de descenso deve reduzir o valor de f, portanto:

$$0 > f(x+d) - f(x) \approx \nabla f(x)^T d \tag{16.15}$$

Concluímos então que uma direção de descenso d deve satisfazer as equações (16.14) e (16.15). A Figura 16.3 ilustra uma direção d que satisfaz ambas as condições. A condição necessária de otimaliade para o problema P_4 é que não exista d satisfazendo as equações (16.14) e (16.15) simultâneamente.

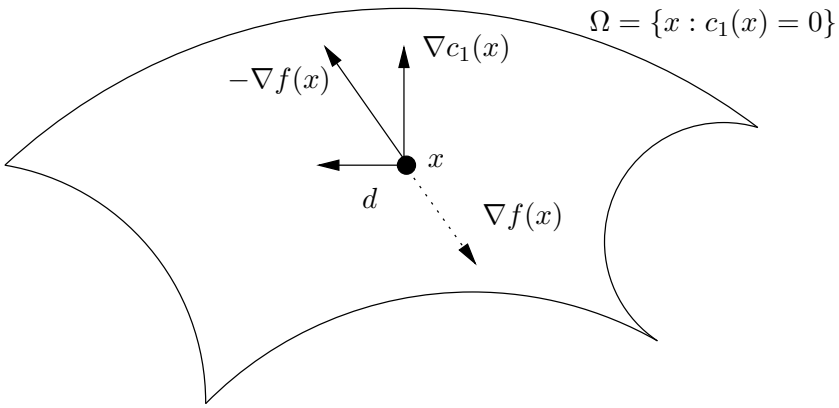


Figura 16.3: Direções de descenso

A única possibilidade para que não exista d satisfazendo as equações (16.14) e (16.15) é que $\nabla f(x) = \lambda_1 \nabla c_1(x)$, ou seja, o gradiente de f no ponto x deve ser linearmente dependente de $\nabla c_1(x)$.

Embora a condição (16.12) pareça ser necessária para que x seja uma solução ótima do problema P_4, ela não é suficiente. Por exemplo, o ponto $x = (1, 1)$ com $\lambda_1 = \frac{1}{2}$ satisfaz a condição $\nabla f(x) = \lambda_1 \nabla c_1(x)$, visto que $\nabla f(x) = [1, 1]^T$ e $\nabla c(x) = [2, 2]^T$. Mas este ponto não é uma solução—na verdade, este ponto maximiza o valor de f sobre o cí rculo.

16.2.2 Uma Restrição de Desigualdade

Considere uma alteração simples de P_4, obtida ao substituir-se a restrição de igualdade por uma desiguladade:

$$\begin{array}{lll} P_5: & \text{Minimize} & x_1 + x_2 \\ & \text{Sujeito a}: & \\ & & 2 - x_1^2 - x_2^2 \geq 0 \end{array} \tag{16.16}$$

A região factí vel é dada na Figura 16.4.

Podemos observar que o ponto $x^* = (-1, -1)$ continua sendo uma solução ótima, como no caso da restrição de igualdade. Se x não é um ponto ótimo, então devemos ser capazes de encontrar um passo d que, ao mesmo tempo, retenha factibilidade e decresça o valor da função objetivo. A direção d decresce o valor de f se $\nabla f(x)^T d < 0$. Entretanto, para manter factibilidade, d deve satisfazer:

$$0 \leq c_1(x+d) \approx c_1(x) + \nabla c_1(x)^T d \Rightarrow c_1(x) + \nabla c_1(x)^T d \geq 0$$

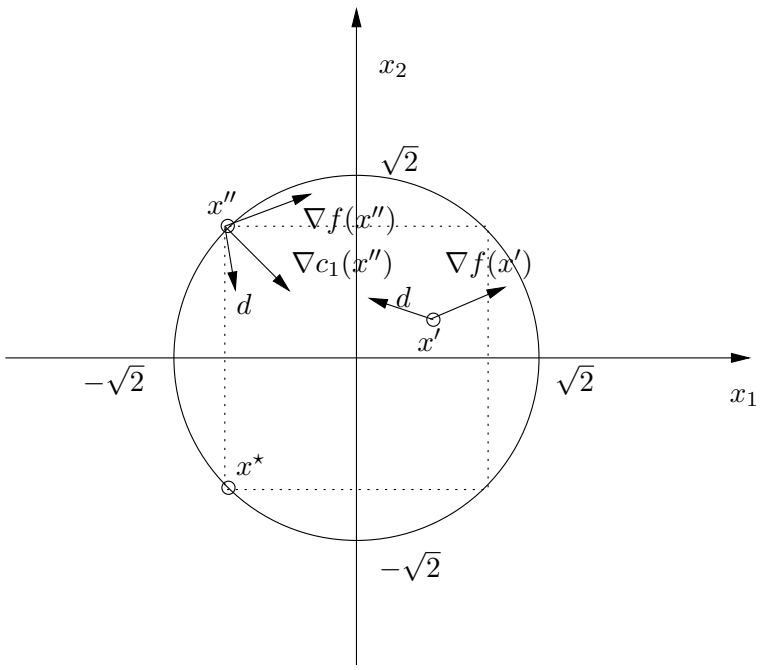


Figura 16.4: Restrição de desigualdade

Ao considerarmos uma direção d que satisfaz ambas as condições, ou seja,

$$\nabla f(x)^T d < 0 \tag{16.17}$$

$$c_1(x) + \nabla c_1(x)^T d \geq 0 \tag{16.18}$$

podemos levar em conta dois casos.

Caso I: x está estritamente dentro do círculo, $c_1(x) > 0$

Neste caso, qualquer vetor d satisfaz a condição (16.18) se $\|d\|$ for pequeno o suficiente. Em particular, sempre que $\nabla f(x) \neq 0$, a direção $d = -c_1(x).\frac{\nabla f(x)}{\|\nabla f(x)\|}$ satisfaz (16.17) e (16.18). A única situação onde esta direção falha é quando $\nabla f(x) = 0$.

Caso II: x está na fronteira do círculo, de forma que $c_1(x) = 0$

Portanto, as condições (16.17) e (16.18) se tornam:

$$\nabla f(x)^T d < 0 \tag{16.19}$$

$$\nabla c_1(x)^T d \geq 0 \tag{16.20}$$

A primeira das condições (16.19) define um meio-espaço aberto, enquanto que a segunda (16.20) define um meio-espaço fechado, como ilustrado na Figura 16.5.

Já a Figura 16.6 ilustra uma região vazia. Pela figura, é fácil de verificar que as duas regiões, $R_1 = \{d : \nabla f(x)^T d < 0\}$ e $R_2 = \{d : \nabla c_1(x)^T d \geq 0\}$, não se interceptam apenas quando $\nabla f(x)$ e $\nabla c_1(x)$ apontam na mesma direção, ou seja,

$$\nabla f(x) = \lambda_1 \nabla c_1(x) \quad \text{para algum } \lambda_1 \geq 0 \tag{16.21}$$

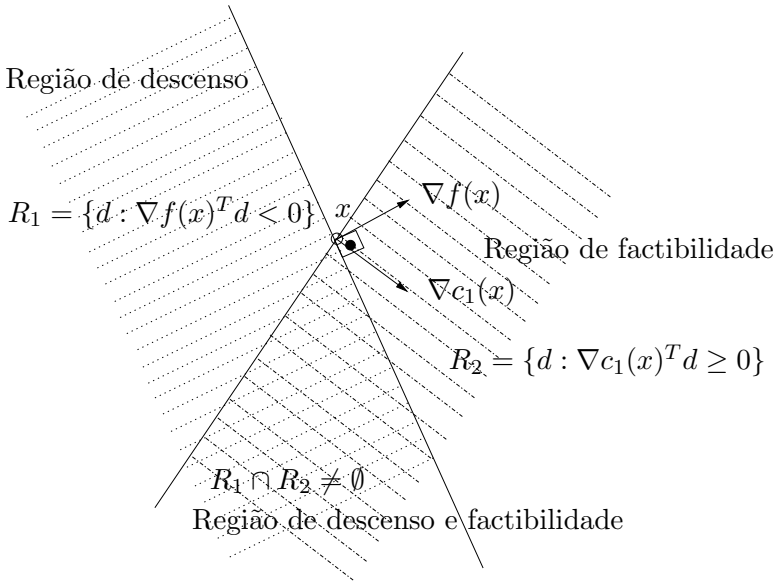

Região de descenso
$R_1 = \{d : \nabla f(x)^T d < 0\}$
x
$\nabla f(x)$
Região de factibilidade
$\nabla c_1(x)$
$R_2 = \{d : \nabla c_1(x)^T d \geq 0\}$
$R_1 \cap R_2 \neq \emptyset$
Região de descenso e factibilidade

Figura 16.5: Região de descenso não vazia

Note que o sinal do multiplicador λ_1 importa. No caso anterior (restrição de igualdade), aceitávamos λ_1 negativo, permitindo que $\nabla f(x)$ e $\nabla c_1(x)$ apontassem em direções opostas.

Considerações Finais

As condições dos casos I e II podem ser expressas de uma forma simplificada, usando a função Lagrangeana:

$$\mathcal{L}(x, \lambda_1) = f(x) - \lambda_1 c_1(x),\ \lambda_1 \geq 0 \tag{16.22}$$

$$\nabla_x \mathcal{L}(x, \lambda_1) = \nabla f(x) - \lambda_1 \nabla c_1(x) = 0 \tag{16.23}$$

As condições ficam, portanto, definidas como:

$$\begin{cases} \nabla f(x) = \lambda_1 \nabla c_1(x) \quad \text{para algum } \lambda_1 \geq 0 \\ \lambda_1 c_1(x) = 0 \end{cases} \tag{16.24}$$

As condições (16.24) implicam que o multiplicador Lagrangeano λ_1 pode ser estritamente positivo apenas quando a restrição correspondente c_1 está ativa. No caso I, temos $c_1(x^*) > 0$ e a condição (16.24) se reduz a $\lambda_1 = 0$ e $\nabla f(x) = 0$, como esperado. No caso II, λ_1 pode assumir qualquer valor não negativo, portanto (16.24) se torna (16.21).

16.2.3 Duas Desigualdades

Adicionamos uma restrição ao problema P_5 e obtemos:

$$\begin{array}{lll} P_6: & \text{Minimize} & x_1 + x_2 \\ & \text{Sujeito a :} & \\ & & 2 - x_1^2 - x_2^2 \geq 0 \\ & & x_2 \geq 0 \end{array} \tag{16.25}$$

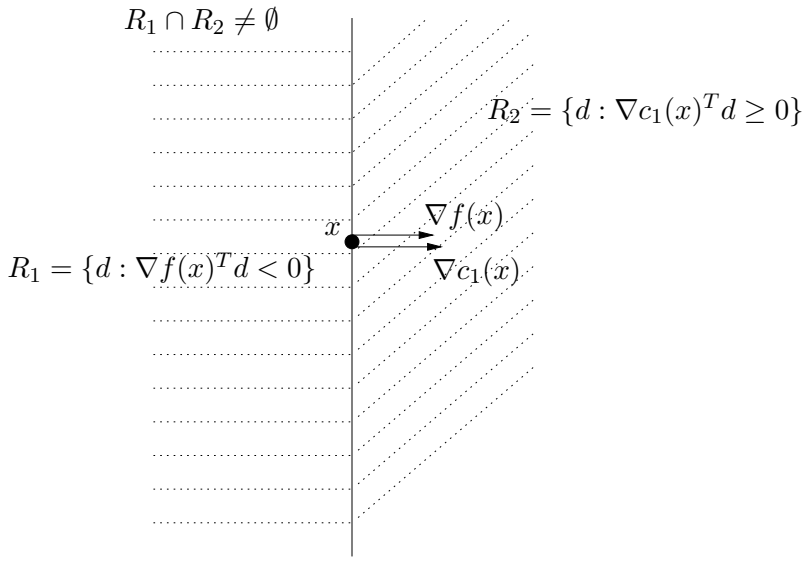


Figura 16.6: Região de descenso vazia

para o qual a região factível é o meio-disco ilustrado na Figura 16.7.

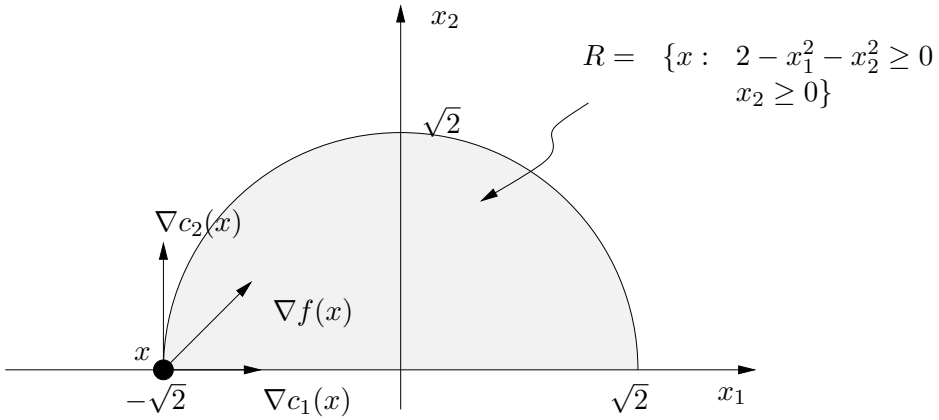


Figura 16.7: Meio disco

É fácil verificarmos que a solução ótima é $(-\sqrt{2}, 0)$, um ponto onde ambas as restrições estão ativas. Usando os argumentos da seção anterior, podemos concluir que d é uma direção de descenso, de primeira ordem, se as seguintes condições são satisfeitas:

$$\nabla c_i(x)^T d \geq 0 \ , \ i \in \mathcal{I} \tag{16.26}$$

$$\nabla f(x)^T d < 0 \tag{16.27}$$

A partir da Figura (16.8), podemos verificar que nenhuma direção d satisfaz as equações (16.26) e (16.27) simultânemamente no ponto $x = (-\sqrt{2}, 0)$.

Vamos definir o Lagrangeano para o problema P_5:

$$\mathcal{L}(x, \lambda) = f(x) - \lambda_1 c_1(x) - \lambda_2 c_2(x)$$

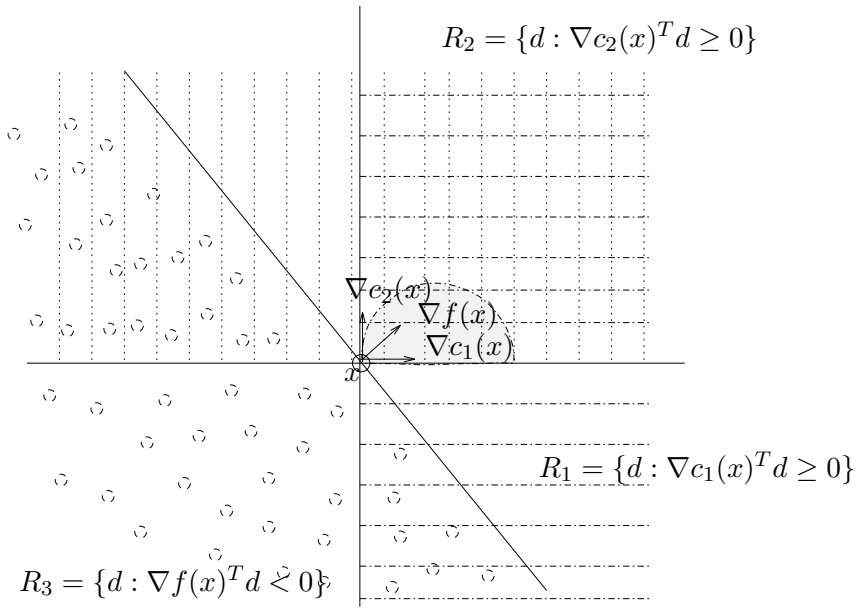


Figura 16.8: Região de descenso vazia

onde $\lambda = (\lambda_1, \lambda_2)$ é um vetor com os multiplicadores de Lagrange. Quando nenhuma direção de descenso de primeira ordem existe no ponto x^*, temos que:

$$\nabla_x \mathcal{L}(x^*, \lambda^*) = 0 \quad \text{para algum } \lambda^* \geq 0 \tag{16.28}$$

Ao aplicarmos a condição de complementaridade a ambas as restrições, obtemos:

$$\lambda_1^* c_1(x^*) = 0 \quad \text{e} \quad \lambda_2^* c_2(x^*) = 0 \tag{16.29}$$

Para $x^* = (-\sqrt{2}, 0)$ temos que:

$$\nabla f(x^*) = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \ \nabla c_1(x^*) = \begin{bmatrix} 2\sqrt{2} \\ 0 \end{bmatrix}, \ \nabla c_2(x^*) = \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

Portanto, é fácil verificar que $\nabla_x \mathcal{L}(x^*, \lambda^*) = 0$ quando selecionamos λ^* como:

$$\lambda^* = \begin{bmatrix} \frac{1}{2\sqrt{2}} \\ 1 \end{bmatrix}$$

Vamos agora examinar o comportamento do Lagrangeano em outros pontos. Para $x = (\sqrt{2}, 0)$, novamente ambas as restrições estão ativas. Contudo, $\nabla f(x)$ não está na região definida por $R = \{d : \nabla c_i(x)^T d \geq 0 \ , \ i = 1, 2\}$. Ver Figura 16.9.

O ponto $d = (-1, 0)$ é uma direção de descenso de primeira ordem, a qual satisfaz as condições (16.26) e (16.27). Note que:

$$\nabla f(x) = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \qquad \nabla c_1(x) = \begin{bmatrix} -2x_1 \\ -2x_2 \end{bmatrix} = \begin{bmatrix} -2\sqrt{2} \\ 0 \end{bmatrix}, \qquad \nabla c_2(x) = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

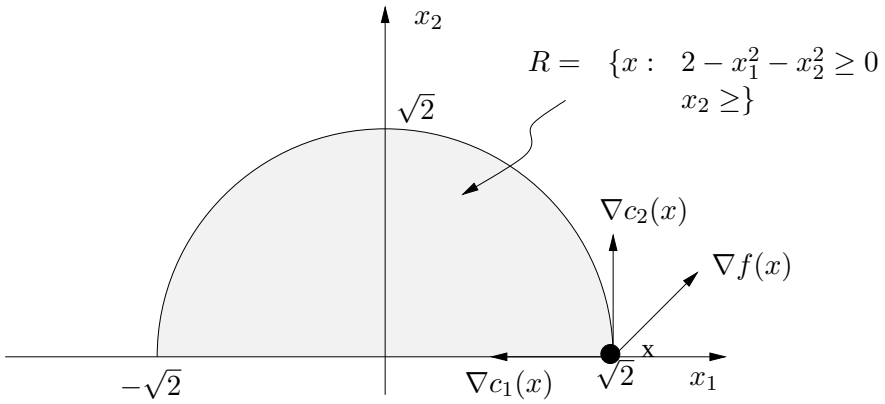


Figura 16.9: Ambas as restrições

Para $x = (1, 0)$, apenas a restrição c_2 está ativa. Linearizando as restrições e a função objetivo, uma direção de descenso deve satisfazer as seguintes desigualdades:

$$\begin{cases} c_1(x) + \nabla c_1(x)^T d \geq 0 \\ c_2(x) + \nabla c_2(x)^T d \geq 0 \\ \nabla f(x)^T d < 0 \end{cases} \Rightarrow \begin{cases} 1 + \nabla c_1(x)^T d \geq 0 \\ \nabla c_2(x)^T d \geq 0 \\ \nabla f(x)^T d < 0 \end{cases} \tag{16.30}$$

Não precisamos nos preocupar com a primeira desigualdade, pois podemos multiplicar d por uma constante pequena até que $1 + \nabla c_1(x)^T d \geq 0$.

Observando que,

$$\nabla f(x) = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \quad \text{e} \quad \nabla c_2(x) = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

é fácil verificar que o vetor $d = (-\frac{1}{2}, \frac{1}{4})$ satisfaz (16.30) e portanto é uma direção de descenso.

16.3 Condições de Otimalidade de Primeira Ordem

Os três exemplos anteriores sugerem que um número de condições são importantes na caracterização de uma solução para o problema (16.1):

a) $\nabla_x \mathcal{L}(x, \lambda) = 0$

b) $\lambda_i \geq 0$ para toda a desigualdade $c_i(x)$

c) Condição de complementaridade $\lambda_i c_i(x) = 0$

Em geral, o Lagrangeano de um problema de otimização com restrições é definido como:

$$\mathcal{L}(x, \lambda) = f(x) - \sum_{i \in \mathcal{E} \cup \mathcal{I}} \lambda_i c_i(x) \tag{16.31}$$

O conjunto ativo $\mathcal{A}(x)$ para qualquer ponto factível x é a união do conjunto $\mathcal{E}$ com os índices das desigualdades ativas, ou seja:

$$\mathcal{A}(x) = \mathcal{E} \cup \{i \in \mathcal{I} : c_i(x) = 0\}$$

O vetor $\nabla c_i(x)$ é tipicamente chamado de *normal* à restrição c_i no ponto x, pois este vetor é usualmente perpendicular à fronteira de c_i no ponto x. É possí vel, entretanto, que $\nabla c_i(x)$ desapareça devido à representação algébrica de c_i, de forma que o termo $\lambda_i \nabla c_i(x)$ se anula para qualquer valor de λ_i e, assim, não desempenha nenhuma função no gradiente do Lagrangeano. Por exemplo, suponha que a restrição de igualdade em P_4 (16.11) seja substituída por:

$$c_1(x) = (x_1^2 + x_2^2 - 2)^2 = 0$$

Neste caso, teríamos $\nabla c_1(x) = 0$ para qualquer ponto factível x. Em particular, a condição $\nabla f(x) = \lambda_1 \nabla c_1(x)$ não é mais válida no ponto ótimo $x^* = (-1, -1)$.

Tipicamente assumimos que as restrições satisfazem uma condição chamada *qualificação de restrição*, que garante que o comportamento degenerado visto acima não ocorrerá. Uma condição de qualificação de restrição é dada abaixo.

Definição 16.1 *Dados um ponto x^* e seu conjunto ativo $\mathcal{A}(x^*)$, dizemos que a condição* "linear independence constraint qualification" *(LICQ) é satisfeita se os gradientes das restrições ativas $\{\nabla c_i(x^*) : i \in \mathcal{A}(x^*)\}$ são todos linearmente independentes.*

Note que se LICQ é satisfeita, nenhum gradiente de uma restrição ativa pode assumir valor 0. Com base na condição LICQ, podemos definir condições necessárias de otimalidade para o problema geral de otimização não-linear sob restrições:

$$\begin{array}{ll} \text{Minimize} & f(x) \\ \text{Sujeito a :} & \\ & c_i(x) = 0 \quad i \in \mathcal{E} \\ & c_i(x) \geq 0 \quad i \in \mathcal{I} \\ & x \in \mathbb{R}^n \end{array} \tag{16.32}$$

As condições a serem dadas abaixo formam a base de algoritmos a serem vistos nos próximos capítulos. Elas são chamadas de *condições de primeira ordem*, pois consideram as propriedades dos gradientes da função objetivo e das restrições.

Teorema 16.1 *(Condições necessárias de primeira ordem) Suponha que x^* é uma solução local para (16.32) e que a condição LICQ é satisfeita por x^*. Então existem multiplicadores de Lagrange λ^*, com componentes λ_i^*, $i \in \mathcal{E} \cup \mathcal{I}$, tal que as seguintes condições são satisfeitas no ponto (x^*, λ^*):*

$$\begin{array}{rcll} \nabla_x \mathcal{L}(x^*, \lambda^*) & = & 0 & \\ c_i(x^*) & = & 0 & \textit{para todo } i \in \mathcal{E} \\ c_i(x^*) & \geq & 0 & \textit{para todo } i \in \mathcal{I} \\ \lambda_i^* & \geq & 0 & \textit{para todo } i \in \mathcal{I} \\ \lambda_i^* c_i(x^*) & = & 0 & \textit{para todo } i \in \mathcal{E} \cup \mathcal{I} \end{array} \tag{16.33}$$

As condições (16.33) são conhecidas como *Karush-Kuhn-Tucker Conditions* (*"KKT Conditions"*). Uma vez que a condição de complementaridade, $\lambda_i^* c_i(x^*) =$

0 para todo $i \in \mathcal{E} \cup \mathcal{I}$, implica que os multiplicadores de Lagrange associados às restrições inativas devem ser nulos, podemos reescrevê-la como:

$$0 = \nabla_x \mathcal{L}(x^*, \lambda^*) = \nabla f(x^*) - \sum_{i \in \mathcal{A}(x^*)} \lambda_i^* \nabla c_i(x^*) \tag{16.34}$$

Observação: Para um dado problema (16.32), podem existir vários vetores λ^* satisfazendo (16.33). Todavia, quando a condição LICQ é satisfeita, a solução λ^* é única.

16.3.1 Exemplo

Considere o problema abaixo:

$$\begin{array}{ll} \text{Minimize} & (x_1 - \frac{3}{2})^2 + (x_2 - \frac{1}{2})^4 \\ \text{Sujeito a :} & \\ & \begin{bmatrix} 1 & -x_1 & -x_2 \\ 1 & -x_1 & x_2 \\ 1 & x_1 & -x_2 \\ 1 & x_1 & x_2 \end{bmatrix} \geq 0 \end{array} \tag{16.35}$$

A região factível é ilustrada na Figura 16.10.

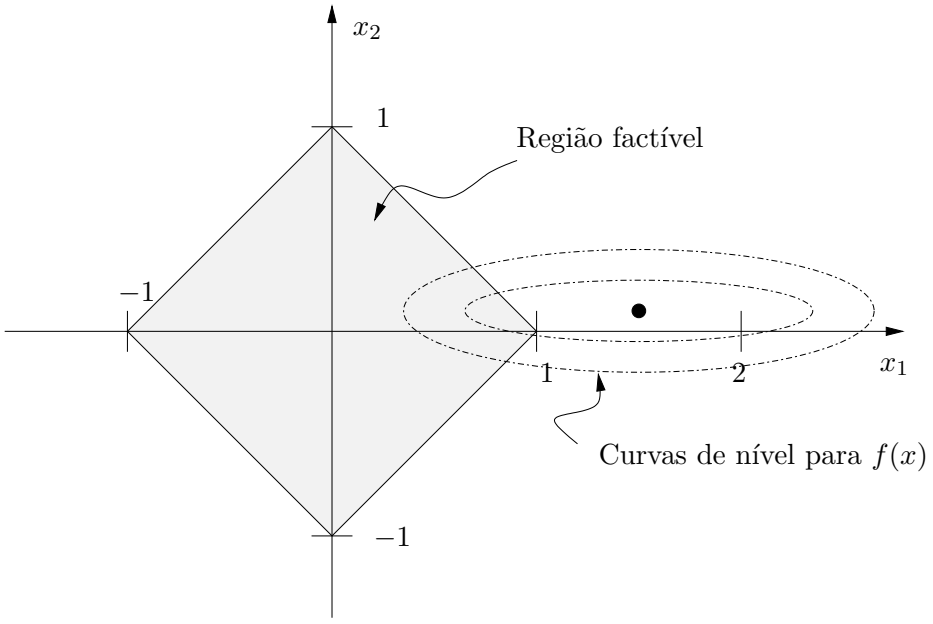


Figura 16.10: Região factível

Com base na figura, podemos nos convencer que a solução ótima é $x^* = (1, 0)$. Apenas a primeira e a segunda desigualdades estão ativas em x^*, $\mathcal{A}(x^*) = \{1, 2\}$.

$$\nabla f(x^*) = \begin{bmatrix} -1 \\ -\frac{1}{2} \end{bmatrix}, \nabla c_1(x^*) = \begin{bmatrix} -1 \\ -1 \end{bmatrix}, \nabla c_2(x^*) = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$$

Portanto, as condições KKT são satisfeitas para $\lambda^* = (\frac{3}{4}, \frac{1}{4}, 0, 0)$.

16.4 Exercícios

EX 16.1: Use as condições KKT para mostrar que o problema abaixo possui um número infinito de ótimos locais:

$$\begin{array}{ll} \text{Minimize} & \|x\|^2 \\ \text{Sujeito a :} & \\ & \|x\|^2 \geq 1, \text{ onde } \| \bullet \| \text{ é a norma Euclidiana.} \end{array}$$

EX 16.2: O problema abaixo possui um número finito ou infinito de ótimos locais? Use as condições KKT para justificar a sua resposta.

$$\begin{array}{ll} \text{Minimize} & (x_2 + 100)^2 + 0.01{x_1}^2 \\ \text{Sujeito a :} & \\ & x_2 - \cos(x_1) \geq 0 \end{array}$$

EX 16.3: Quais são as dificuldades para se resolver o problema abaixo com software de otimização?

$$\begin{array}{ll} \text{Minimize} & |x| + y^2 \\ x, y & \\ \text{Sujeito a :} & x^2 = 1 \end{array}$$

Proponha uma formulação alternativa equivalente para o problema.

EX 16.4: Quais são as dificuldades para se resolver o problema abaixo com software de otimização?

$$\begin{array}{ll} \text{Minimize} & \text{Max } \{|y_k - d_k| : k = 1, 2, 3\} \\ x_1, x_2 & \end{array}$$

onde $y_k = x_1 + x_2 k$, $k = 1, 2, 3$, e $d = (1, 1.5, 1.2)$. Proponha uma formulação alternativa equivalente para o problema.

EX 16.5: Seja $f : \mathbb{R}^n \to \mathbb{R}^n$ uma função não-linear contínua e diferencialmente contínua. Prof. Kunz dispõe de um pacote de otimização de funções não-lineares contínuas que resolve problemas da forma:

$$\begin{array}{ll} \text{Minimize} & g(x) \\ x \in \mathbb{R}^n & \end{array}$$

onde $g : \mathbb{R}^n \to \mathbb{R}$ é uma função contínua e diferenciável. Prof. Kunz afirma que se pode utilizar o pacote de software para encontrar uma raiz para f, ou seja, $x \in \mathbb{R}^n$ tal que $f(x) = 0$. Você concorda ou discorda do Prof. Kunz? Se você discorda, mostre porque não podemos utilizar o pacote de software. Se você concorda, mostre como que uma raiz pode ser encontrada.

EX 16.6: Seja $f : \mathbb{R} \to \mathbb{R}$ uma função não-linear contínua e diferencialmente contínua. Prof. Kunz dispõe de um pacote para solução de sistemas de equações e desigualdades não-lineares, que encontra, caso exista, uma solução x para:

$$\begin{array}{l} g(x) < 0 \\ h(x) = 0 \end{array}$$

onde $g : \mathbb{R} \to \mathbb{R}$ e $h : \mathbb{R} \to \mathbb{R}$ são funções contínuas e diferenciáveis. Prof. Kunz afirma que se pode utilizar o pacote de software para encontrar um mínimo local para $f(x)$. Você concorda ou discorda do Prof. Kunz? Se você discorda, mostre porque não podemos utilizar o pacote de software. Se você concorda, mostre como que um mínimo local pode ser encontrado.

EX 16.7: Desenvolva as condições KKT para o problema abaixo e obtenha as equações que devem, necessariamente, ser satisfeitas por um mínimo local:

$$\begin{aligned} \text{Minimize} \quad & \sum_{i=1}^{n} c_i x_i + \sum_{i=1}^{n} g_i(x_i) \\ \text{Sujeito a :} \quad & \\ & \sum_{i=1}^{n} x_j \leqslant b \\ & x_i \geqslant 0, \ i = 1, \ldots, n \end{aligned}$$

onde $g_i(x_i), i = 1, \ldots, n$, são funções diferenciáveis.

Capítulo 17

Programação Não-Linear: Fundamentos de Algoritmos

Aqui iniciamos a discussão de algoritmos para a solução do problema geral de otimização sob restrições.

$$\begin{array}{lll} P: & Minimize & f(x) \\ & x \in \mathbb{R}^n & \\ & Sujeito\ a: & \\ & & c_i(x) = 0 \quad i \in \mathcal{E} \\ & & c_i(x) \leq 0 \quad i \in \mathcal{I} \end{array} \tag{17.1}$$

Onde, a função objetivo f e as restrições c_i são funções suaves com valores reais, $\mathcal{I}$ e $\mathcal{E}$ são conjuntos de índices finitos para as restrições de desigualdade e igualdade, respectivamente. No capítulo anterior desenvolvemos condições que caracterizam as soluções ótimas (*Karush-Kuhn-Tucker Conditions*). Algoritmos, para serem eficientes, devem levar em conta propriedades e a estrutura das funções que aparecem em (17.1). Algumas classes de problemas, para os quais existem algoritmos especializados, são:

a) Programação Linear

b) Programação Quadrática

c) Programação Não-Linear

A grande maioria dos algoritmos para otimização não-linear é iterativa: eles geram uma seqüência x^k de soluções aproximadas que esperamos que seja convergente a uma solução ótima local x^*. Os algoritmos utilizam informações sobre a solução corrente x^k, bem como as derivadas de f e c_i, $i \in \mathcal{E} \cup \mathcal{I}$, no processo de obtenção do próximo iterando, x^{k+1}.

Estudo Inicial de um Problema

É recomendável o estudo inicial de um problema, no sentido de que simplificações podem ser encontradas:

a) Um exame das restrições pode indicar que a região factível é vazia;

b) A função objetivo pode ser ilimitada;

c) Talvez seja possível identificar quais restrições estarão ativas na solução ótima, permitindo o emprego direto das condições KKT.

Um aspecto relevante durante a modelagem e a escolha de um algoritmo se refere aos tipos de restrições:

Restrições Duras Aquelas que devem ser satisfeitas para que os resultados sejam significativos, e.g. a restrição de que $x \geq 0$ para que uma outra restrição tal como $\sqrt{x} \leq 2$ tenha significado; as restrições duras podem até mesmo não ser definidas fora da região de factibilidade.

Restrições Flexíveis Aquelas que são definidas mesmo na região infactível.

17.1 Categorizando Algoritmos de Otimização

17.1.1 O Método da Função Penalidade

Podemos atacar o problema (17.1) por meio da combinação da função objetivo e das restrições em uma função penalidade, e resolvendo uma seqüência de problemas irrestritos. Se apenas restrições de igualdade estão presentes, isto é ($\mathcal{I} = \phi$), então a função de penalidade pode tomar a forma:

$$f(x) + \frac{1}{2\mu} \sum_{i \in \mathcal{E}} c_i^2(x) \tag{17.2}$$

Onde $\mu > 0$ é chamado de parâmetro de penalidade. Minimizamos a função acima, sem restrições, para uma série decrescente de valores de μ , até que uma solução do problema original seja encontrada.

17.1.2 O Método de Barreiras

Adicionamos à função objetivo, termos que são insignificantes quando x está seguramente no interior da região factível mas que tendem a $+\infty$ quando x se aproxima da fronteira da região factível. Se apenas desigualdades estão presentes, a função de barreira logarítmica tem forma:

$$f(x) - \mu \sum_{i \in \mathcal{I}} \log c_i(x) \tag{17.3}$$

Onde $\mu > 0$ é dito parâmetro de barreira.

O minimizador da função (17.3) tende para uma solução do problema original à medida que μ se aproxima de zero, sob certas condições. Novamente, a estratégia é encontrar um minimizador aproximado para uma seqüência decrescente de valores de μ.

17.1.3 O Método Lagrangeano Aumentado

Neste método procura-se combinar as propriedades da função Lagrangeana (17.4) e da penalidade quadrática (17.2).

$$\mathcal{L}(x, y) = f(x) - \sum_{i \in \mathcal{E} \cup \mathcal{I}} \lambda_i c_i(x), \quad \lambda_i \geq 0,\ i \in \mathcal{I} \tag{17.4}$$

Quando o problema (17.2) possui apenas restrições de igualdade, a função Lagrangeana aumentada toma a forma:

$$\mathcal{L}(x, \lambda, \mu) = f(x) - \sum_{i \in \mathcal{E}} \lambda_i c_i(x) + \frac{1}{2\mu} \sum_{i \in \mathcal{E}} c_i^2(x) \tag{17.5}$$

Métodos baseados na função (17.5) procedem fixando λ a alguma estimativa dos multiplicadores de Lagrange ótimos e μ a um valor positivo, depois encontram um valor x que aproximadamente minimize $\mathcal{L}(x, \lambda, \mu)$. O iterando x é utilizado para recalcular o valor de λ, permitindo que μ talvez seja decrescido.

17.1.4 Método Seqüencial Linear

Esses métodos minimizam, a cada iteração uma certa função Lagrangeana sujeito a uma linearização das restrições.

17.1.5 Programação Quadrática Seqüencial

A idéia aqui é modelar o problema (17.1) como um subproblema quadrático a cada iteração e definir uma direção de busca como solução deste subproblema. Mais especificamente, no caso de existirem apenas restrições de igualdade, definimos a direção de busca p_k a cada iterando (x_k, λ_k) como a solução do problema abaixo.

$$\begin{array}{ll} Minimize & \frac{1}{2} p^T W_k p + \nabla f_k^T p \\ p & \\ Sujeito\ a: & \\ & A_k p + c_k = 0 \end{array} \tag{17.6}$$

O objetivo do problema é uma aproximação da função Lagrangeana e as restrições são uma linearização das restrições do problema (17.1). O iterando seguinte é obtido por meio de uma busca ao longo da direção p_k, obtida a partir da solução de (17.6), até que uma certa função mérito seja decrescida. Programação Quadrática Seqüencial é considerado um dos métodos mais eficientes na prática.

17.2 Eliminação de Variáveis

Uma abordagem natural é tentar eliminar restrições de maneira a se obter um problema irrestrito, ou pelo menos eliminar algumas restrições e se obter um problema

simplificado. Abaixo, ilustramos o procedimento de eliminação de restrições em um exemplo simples:

$$\begin{array}{ll} Minimize & f(x) = f(x_1, x_2, x_3, x_4) \\ Sujeito\ a: & \\ & x_1 + x_3^2 - x_3 x_4 = 0 \\ & -x_2 + x_4 + x_3 = 0 \end{array} \tag{17.7}$$

Não há risco algum em definirmos:

$$\left\{ \begin{array}{rcrcl} x_1 & = & x_3 x_4 & - & x_3^2 \\ x_2 & = & x_4 & + & x_3^2 \end{array} \right. \tag{17.8}$$

E trabalhamos com o problema irrestrito:

$$\begin{array}{ll} Minimize \\ x_3, x_4 \end{array} \quad f(x_3 x_4 - x_3^2, x_4 + x_3^2, x_3, x_4) \tag{17.9}$$

O problema (17.9) pode ser resolvido com os métodos estruturados no início da disciplina.

17.2.1 Exemplo: Problemas Que Podem Surgir

Considere o problema:

$$\begin{array}{ll} Minimize & x^2 + y^2 \\ Sujeito\ a: & \\ & (x-1)^3 = y^2 \end{array} \tag{17.10}$$

A Figura 17.1 mostra as curvas de nível para a função objetivo e a restrição, a qual mostra que a solução ótima é $(x, y) = (1, 0)$.

Considere a tentativa de eliminarmos a variável y e resolvermos o problema irrestrito abaixo:

$$Mininize\ h(x) = x^2 + (x-1)^3 \tag{17.11}$$

Claramente $h(x) \to -\infty$ se $x \to -\infty$. **O que aconteceu?**

Ao eliminarmos a restrição ignoramos que a restrição $(x-1)^3 = y^2$ implicitamente impõe a restrição $x \geq 1$ que está ativa na solução ótima. Este exemplo ilustra que a eliminação de restrições não-lineares pode resultar em erros que são difíceis de se identificar.

17.2.2 Eliminação de Restrições Lineares

Vamos considerar a minimização de uma função não-linear sujeita a um conjunto de igualdades lineares.

$$\begin{array}{ll} Minimize & f(x) \\ Sujeito\ a: & \\ & Ax = b \end{array} \tag{17.12}$$

Onde $A \in \mathbb{R}^{m \times n}$ e $m \leq n$. Suponha que $rank(A) = m$. Sob essas condições, podemos encontrar um subconjunto de m colunas de A que são linearmente independentes.

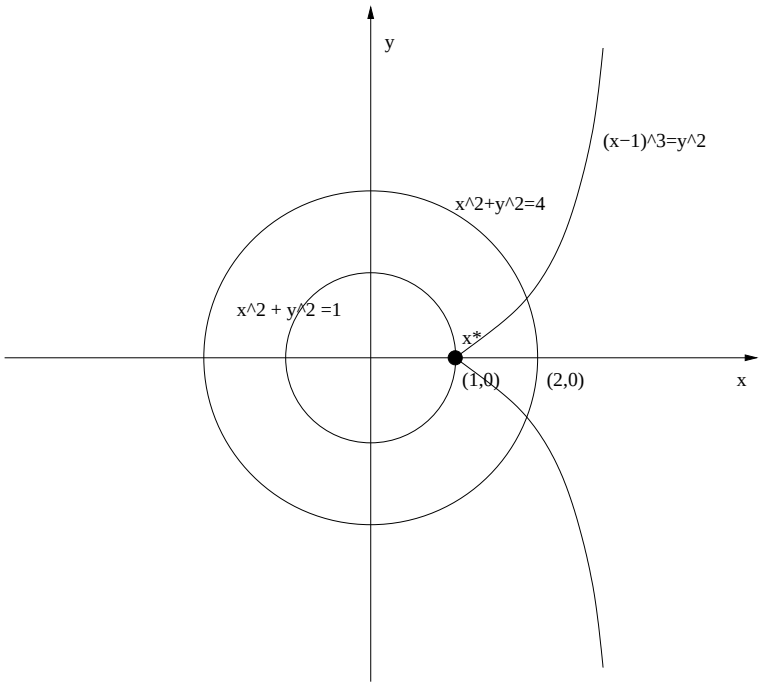


Figura 17.1: Curvas de nível das funções $x^2 + y^2$ e $y^2 = (x-1)^3$

Podemos reunir essas colunas linearmente independentes na matriz B, definindo uma matriz $P \in \mathbb{R}^{n\times n}$ de permutação que move as colunas linearmente independentes para as primeiras colunas de A. Em outras palavras,

$$AP = \left[\begin{array}{c|c} B & N \end{array}\right] \tag{17.13}$$

onde N denota as $n-m$ colunas restantes.

Definimos os subvetores $x_B \in \mathbb{R}^m$ e $x_N \in \mathbb{R}^{n-m}$ de forma que:

$$P^T x = \left[\begin{array}{c} x_B \\ x_N \end{array}\right] \tag{17.14}$$

E dizemos que x_B são as variáveis básicas e x_N as variáveis não-básicas. Notando que $PP^T = I$, podemos reescrever a restrição $Ax = b$ como:

$$\begin{aligned} b &= Ax \\ &= APP^T x \\ &= (AP)(P^T x) \\ &= \left[\begin{array}{c|c} B & N \end{array}\right] \left[\begin{array}{c} x_B \\ x_N \end{array}\right] \\ &= Bx_B + Nx_N \end{aligned} \tag{17.15}$$

A partir dessa fórmula, deduzimos que as variáveis básicas são dadas por:

$$x_B = B^{-1}b - B^{-1}Nx_N \tag{17.16}$$

Podemos então computar um ponto factível para as restrições $Ax = b$ escolhendo qualquer valor de x_N, e então definindo x_B de acordo com a fórmula (17.16). O problema (17.2) é portanto equivalente ao problema irrestrito que segue abaixo:

$$Min \;\; f(x_N) = f(P \begin{bmatrix} B^{-1}b - B^{-1}Nx_N \\ x_N \end{bmatrix}) \tag{17.17}$$

Exemplo: Considere o problema abaixo

$$\begin{array}{ll} Minimize & \sin(x_1 + x_2) + x_3^2 + \frac{1}{3}(x_4 + x_5^4 + \frac{x_6}{2}) \\ Sujeito\ a: & \\ & 8x_1 - 6x_2 + x_3 + 9x_4 + 4x_5 = 6 \\ & 3x_1 + 2x_2 - x_4 + 6x_5 + 4x_6 = -4 \end{array} \tag{17.18}$$

Definimos a matriz P de forma a reordenar os componentes de x como $x^T = (x_3, x_6, x_1, x_2, x_4, x_5)^T$, obtemos:

$$P = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{17.19}$$

$$AP = \begin{bmatrix} 1 & 0 & 8 & -6 & 9 & 4 \\ 0 & 4 & 3 & 2 & -1 & 6 \end{bmatrix} \tag{17.20}$$

A matriz base B é diagonal, portanto fácil de se inverter. Com base em (17.16), obtemos:

$$\begin{bmatrix} x_3 \\ x_6 \end{bmatrix} = - \begin{bmatrix} 8 & -6 & 9 & 4 \\ \frac{3}{4} & \frac{1}{2} & \frac{-1}{4} & \frac{3}{2} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} 6 \\ -1 \end{bmatrix} \tag{17.21}$$

Substituindo-se x_3 e x_6 em (17.18), obtemos o seguinte problema irrestrito:

$$\begin{array}{ll} Minimize & f(x_1, x_2, x_4, x_5) = \sin(x_1 + x_2) + (8x_1 - 6x_2 + 9x_4 + 4x_5 - 6)^2 \\ x_1, x_2, x_4, x_5 & \quad + \frac{1}{3}\left[x_4 + x_5^4 - (\frac{1}{2} + \frac{3}{8}x_1 + \frac{1}{4}x_2 - \frac{1}{8}x_4 + \frac{3}{4}x_5)\right] \end{array} \tag{17.22}$$

17.2.3 Os Efeitos das Desigualdades

Eliminação de variáveis não é sempre recomendada na presença de igualdades e desigualdades. Considere o problema a seguir:

$$\begin{array}{ll} Minimize & \sin(x_1 + x_2) + x_3^2 + \frac{1}{3}(x_4 + x_5^2 + \frac{x_6}{2}) \\ Sujeito\ a: & \\ & 8x_1 - 6x_2 + x_3 + 9x_4 + 4x_5 = 6 \\ & 3x_1 + 2x_2 - x_4 + 6x_5 + 4x_6 = -4 \\ & x_1, x_2, x_3, x_4, x_5, x_6 \geq 0 \end{array} \tag{17.23}$$

Após eliminarmos as variáveis x_3 e x_6, conforme desenvolvimento anteiro, obteremos o seguinte conjunto de restrições:

$$\begin{array}{rcl} (x_1, x_2, x_4, x_5) & \geq & 0 \\ 8x_1 - 6x_2 + 9x_4 + 4x_5 & \leq & 6 \\ \frac{3}{4}x_1 + \frac{1}{2}x_2 - \frac{1}{4}x_4 + \frac{3}{2}x_5 & \leq & -1 \end{array} \tag{17.24}$$

Portanto, o custo de eliminarmos as igualdades foi tornar as desigualdades muito mais complicadas.

Capítulo 18

Programação Quadrática

Um problema de otimização com função objetivo quadrática e restrições lineares é dito problema de programação quadrática. Por si só esta classe de problemas é relevante mas também desempenha um papel fundamental em métodos gerais de otimização não-linear sob restrições como, por exemplo, programação quadrática seqüêncial.

Na sua forma geral, o problema quadrático é expresso como segue:

$$\begin{array}{lll} P_Q: & \text{Minimize} & q(x) = \frac{1}{2}x^T G x + d^T x \\ & \text{Sujeito a:} & \\ & & a_i^T x = b_i, i \in \mathcal{E} \\ & & a_i^T x \geq b_i, i \in \mathcal{I} \end{array} \tag{18.1}$$

onde G é uma matriz simétrica $n \times n$, $\mathcal{E}$ e $\mathcal{I}$ são conjuntos finitos de índices de restrições, e d, x e a_i, $i \in \mathcal{E} \cup \mathcal{I}$, são vetores coluna com n elementos.

Problemas P_Q podem ser resolvidos em um número finito de iterações, mas o esforço computacional depende significativamente das *características da função objetivo* e do *número de restrições de desigualdade*. Se a matriz Hessiana G é positiva semi-definida, então dizemos que (18.1) é um problema quadrático convexo e, neste caso, a solução do problema é comparável à resolução de um problema de programação linear. Problemas quadráticos não-convexos, quando G é uma matriz indefinida, são muito mais difíceis de serem resolvidos.

Neste capítulo nos concentramos na busca de uma solução para problemas quadráticos convexos (ou pontos estacionários para problemas não-convexos). O capítulo está organizado como segue. A Seção 18.1 apresenta o problema de investimento com maximização dos retornos e minimização dos riscos, o qual pode ser colocado como um problema de programação quadrática. A Seção 18.2 desenvolve uma solução do problema quadrático quando apenas restrições de igualdade estão presentes. A Seção 18.3 discute brevemente as classes de algoritmos para resolver o problema quadrático na presença de desigualdades, bem como condições de otimalidade. A Seção 18.4 apresenta do algoritmo de conjunto ativo que generaliza o algoritmo simplex para programação linear.

18.1 Exemplo: Otimização de Portofolio

Considere um problema de aplicações em fundos de ações cujas informações acerca dos investimentos são:

- Há um conjunto $N = \{1, \dots \ n\}$ de possíveis investimentos.
- O investimento i é modelado por uma variável aleatória r_i com distribuição normal, tendo valor esperado $\mu_i = E[r_i]$ e variância $\sigma_i^2 = E[(r_i - \mu_i)^2]$ conhecidos.
- Quanto maior o valor da variância σ_i^2, maior o risco do investimento i.

O investidor deseja aplicar o montante disponível nos diferentes investimentos sendo x_i a fração a ser aplicada no investimento i. Daí surgem as restrições:

$$x_i \geq 0, \qquad i = 1, \dots, n \tag{18.2}$$

$$\sum_{i=1}^{n} x_i = 1 \tag{18.3}$$

Para medir a qualidade de um plano de investimentos (portofólio), que corresponde ao vetor $x \in \mathbb{R}^n$, devemos obter uma medida do retorno esperado, sendo este dado por:

$$R = \sum_{i=1}^{n} x_i r_i \tag{18.4}$$

Levando em consideração que uma variável aleatória não pode ser otimizada diretamente, podemos medir a qualidade do investimento de acordo com o valor esperado, obtendo portanto a expressão:

$$\begin{aligned} E[R] &= E[\sum_{i=1}^{n} x_i r_i] = \sum_{i=1}^{n} x_i E[r_i] = \sum_{i=1}^{n} x_i \mu_i \\ &= x^T \mu \end{aligned} \tag{18.5}$$

onde $\mu = (\mu_1, \dots, \mu_n)$.

A variância do retorno total esperado pode ser calculada a partir do coeficiente de correlação ρ_{ij} entre pares de variáveis, conforme a expressão:

$$\rho_{ij} = \frac{E[(r_i - \mu_i)(r_j - \mu_j)]}{\sigma_i \sigma_j} \text{ para todo par } i, j \tag{18.6}$$

O coeficiente de correlação ρ_{ij} mede a tendência de duas variáveis aleatórias r_i e r_j tenderem na mesma direção[1]. Dois investimentos que crescem/decrescem juntos têm

[1] $E[(r_i - \mu_i)(r_j - \mu_j)]$ é difinido como a covariância das variáveis aleatórias r_i e r_j. Note que se r_i e r_j são independentes, então $E[(r_i - \mu_i)(r_j - \mu_j)] = E[(r_i - \mu_i)]E[(r_j - \mu_j)] = 0$. Note ainda que se $r_i = r_j$ então $E[(r_i - \mu_i)(r_j - \mu_j)] = E[(r_i - \mu_i)^2] = \sigma_i^2$ e, portanto, $\rho_{ij} = 1$. O coeficiente de correlação consiste da normalização da covariância. A correlação é máxima quando $|\rho_{ij}| = 1$ e mínima quando $\rho_{ij} = 0$.

correlação positiva. À medida que ρ_{ij} se aproxima de 1, mais similares são os investimentos i e j. Investimentos que tendem a se mover em direções opostas apresentam correlação negativa.

Portanto, a variância do investimento total R é dada por :

$$\begin{aligned} E[(R - E[R])^2] &= \sum_{i=1}^{n}\sum_{j=1}^{n} x_i\sigma_i\sigma_j\rho_{ij}x_j \\ &= x^T G x \end{aligned} \tag{18.7}$$

sendo a matriz simétrica $G \in \mathbb{R}^{n\times n}$ definida como:

$$G_{ij} = \sigma_i\sigma_j\rho_{ij} \tag{18.8}$$

A matriz G é positiva semi-definida, todavia não desenvolveremos uma demonstração. Estamos interessados em um portofólio cujo retorno total $x^T\mu$ seja alto ao mesmo tempo que a variância x^TGx seja pequena. No modelo proposto por Markowitz, combina-se estes dois objetivos em uma única função usando o chamado *"parâmetro de tolerância de risco"* denotado por k. Então, resolvemos o problema abaixo.

$$\begin{aligned} &\text{Maximize} \quad x^T\mu - kx^TGx \\ &\text{Sujeito a :} \\ &\qquad \sum_{i=1}^{n} x_i = 1 \\ &\qquad x \geq 0 \end{aligned} \tag{18.9}$$

O problema (18.9) expressa o desejo de maximizar o retorno e simultaneamente minimizar os riscos incorridos nos investimentos. As restrições garantem que o vetor $x \in \mathbb{R}^n$ induz um portofólio.

18.2 Propriedades de Problemas Quadráticos com Apenas Restrições de Igualdade

Considere o problema abaixo:

$$\begin{aligned} P_Q^{EC} : \quad &\text{Minimize} \quad q(x) = \tfrac{1}{2}x^TGx + d^Tx \\ &\text{Sujeito a :} \\ &\qquad Ax = b \end{aligned} \tag{18.10}$$

onde: $x \in \mathbb{R}^n$; $A \in \mathbb{R}^{m\times n}$; $b \in \mathbb{R}^m$, $d \in \mathbb{R}^n$; e $G \in \mathbb{R}^{n\times n}$ tal que $G = G^T$. Assumimos que A tem posto completo, ou seja, $rank(A) = m$. Aplicando as condições de otimalidade de primeira ordem (*KKT Conditions*), deduzimos que um par solução multiplicador-Lagrange (x, λ) deve satisfazer as restrições abaixo:

$$\begin{cases} Gx + d - (\lambda^TA)^T = 0 \\ Ax - b = 0 \end{cases} \tag{18.11}$$

Reescrevendo (18.11) em termos da solução ótima x^* e do vetor com os multiplicadores de Lagrange λ^*, temos:

$$\begin{bmatrix} G & -A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} x^* \\ \lambda^* \end{bmatrix} = \begin{bmatrix} -d \\ b \end{bmatrix} \tag{18.12}$$

Podemos novamente reescrever (18.12) em uma outra forma que facilita a computação. Fazendo $x^* = x + p$ onde x é uma estimativa da solução ótima, obtemos:

$$\begin{bmatrix} G & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} -p \\ \lambda^* \end{bmatrix} = \begin{bmatrix} g \\ c \end{bmatrix} \tag{18.13}$$

onde:

$$\begin{cases} c &= Ax - b \\ g &= d + Gx, \text{ e} \\ p &= x^* - x \end{cases} \tag{18.14}$$

Lema 18.1 *Seja A uma matriz $m \times n$, $m \leq n$, tal que $rank(A) = m$. Seja Z uma matriz que define uma base para $Null(A)$, ou seja, $Null(A) = \{Zu : u \in \mathbb{R}^{n-m}\}$. Se a matriz Hessiana reduzida Z^TGZ é positiva definida, então a matriz KKT:*

$$K = \begin{bmatrix} G & A^T \\ A & 0 \end{bmatrix} \tag{18.15}$$

é não singular e, portanto, o par (x^*, λ^*) satisfazendo (18.12) é único. ■

Prova: Se existe (p, λ) satisfazendo:

$$\begin{bmatrix} G & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} p \\ \lambda \end{bmatrix} = 0$$

Então temos que:

$$0 = \begin{bmatrix} p \\ \lambda \end{bmatrix}^T \begin{bmatrix} G & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} p \\ \lambda \end{bmatrix} = p^TGp$$

Uma vez que $Ap = 0$, segue que $p \in Null(A)$ e $p = Zu$. Portanto:

$$u^TZ^TGZu = 0$$

e daí segue que $u = 0$ pois Z^TGZ tem posto completo (a matriz é positiva definida). Isto nos leva a concluir que $Gp + A^T\lambda = 0 \Leftrightarrow GZu + A^T\lambda = 0 \Leftrightarrow A^T\lambda = 0 \Leftrightarrow \lambda = 0$ pois as colunas de A^T são linearmente independentes. Portanto, a única solução de $Kz = 0$ é $z = 0$ e, consequentemente, K é não singular.

Se (x^*, λ^*) satisfaz (18.12), então:

$$\begin{bmatrix} G & -A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} x^* \\ \lambda^* \end{bmatrix} = \begin{bmatrix} -d \\ b \end{bmatrix} \Leftrightarrow \begin{bmatrix} G & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} x^* \\ -\lambda^* \end{bmatrix} = \begin{bmatrix} -d \\ b \end{bmatrix}$$
$$\Leftrightarrow K \begin{bmatrix} x^* \\ -\lambda^* \end{bmatrix} = \begin{bmatrix} -d \\ b \end{bmatrix}$$

Portanto, o par (x^*, λ^*) é único pois K é não singular. ■

Corolário 18.1 *Se as condições do Lema 18.1 são satisfeitas, então o vetor x^* que satisfaz (18.12) é a solução global única do problema quadrático (18.10).*

Exemplo 1

Considere a instância de P_Q^{EC} dada a seguir:

$$\begin{aligned} \text{Minimize} \quad & q(x) = 3x_1^2 + 2x_1x_2 + x_1x_3 + 2.5x_2^2 \\ & \qquad +2x_2x_3 + 2x_3^2 - 8x_1 - 3x_2 - 3x_3 \\ \text{Sujeito a :} \quad & \\ & x_1 + x_3 = 3 \\ & x_2 + x_3 = 0 \end{aligned} \tag{18.16}$$

Podemos reescrever (18.16) na forma (18.10), fazendo:

$$G = \begin{bmatrix} 6 & 2 & 1 \\ 2 & 5 & 2 \\ 1 & 2 & 4 \end{bmatrix}, \quad d = \begin{bmatrix} -8 \\ -3 \\ -3 \end{bmatrix}, \quad A = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} \quad \text{e} \quad b = \begin{bmatrix} 3 \\ 0 \end{bmatrix} \tag{18.17}$$

A solução ótima x^* e os multiplicadores de Lagrange ótimos λ^* são:

$$x^* = \begin{bmatrix} 2 \\ -1 \\ 1 \end{bmatrix} \quad \text{e} \quad \lambda^* = \begin{bmatrix} 3 \\ -2 \end{bmatrix} \tag{18.18}$$

No exemplo acima, a matriz G é positiva definida e a matriz base do espaço nulo de A pode ser definida como:

$$Z = \begin{bmatrix} -1 & -1 & 1 \end{bmatrix}^T \tag{18.19}$$

18.2.1 Resolvendo o Sistema KKT

Aqui discutiremos como resolver eficientemente o sistema KKT (18.12) ou (18.13). A matriz do sistema (18.12), aqui definida por K, pode ser indefinida e portanto a fatoração de Cholesky pode não ser aplicável. Se K é uma matriz positiva definida, então $K = LL^T$ onde L é uma matriz triangular inferior, o que nos levaria a obtenção da solução única por meio de substituição direta e retro-substituição após o cômputo do fator Cholesky L. Caso K não seja positiva definida, podemos então utilizar a eliminação de Gauss ou a fatoração LU, encontrando matrizes triangulares L e U tal que $K = LU$. Neste último caso podemos também calcular uma solução rapidamente através da retro-substituição e substituição direta. Vale lembrar que toda a matriz quadrada não singular possui uma fatoração LU.

18.3 Problemas Quadráticos Sob Restrições

Existem três grandes classes de algoritmos

1) **Método Clássico de Conjunto Ativo:** pode ser aplicado tanto a problemas convexos quanto não-convexos.

2) **Método de Projeção de Gradiente:** tentativa de acelerar o processo de solução do método de conjunto ativo.

3) **Método de Ponto Interior**

18.3.1 Condições de Otimalidade

Relembrando o problema quadrático, desejamos resolver o problema:

$$\begin{array}{ll} \text{Minimize} & q(x) = \frac{1}{2}x^T Gx + d^T x \\ \text{Sujeito a :} & \\ & a_i^T x = b_i, \quad i \in \mathcal{E} \\ & a_i^T x \geq b_i, \quad i \in \mathcal{I} \end{array} \tag{18.20}$$

Para um ponto ótimo x^* de (18.20), definimos conjunto ativo $\mathcal{A}(x^*)$ como os índices das restrições que são satisfeitas com igualdade, ou seja,

$$\mathcal{A}(x^*) = \{i \in \mathcal{E} \cup \mathcal{I} : a_i^T x^* = b_i\} \tag{18.21}$$

Desenvolvendo as condições KKT para (18.20), obtemos:

$$Gx^* + d - \sum_{i \in \mathcal{A}(x^*)} \lambda_i^* a_i = 0 \tag{18.22}$$

$$a_i^T x^* = b_i, \quad i \in \mathcal{A}(x^*) \tag{18.23}$$

$$a_i^T x^* \geq b_i, \quad i \in \mathcal{I} - \mathcal{A}(x^*) \tag{18.24}$$

$$\lambda_i^* \geq 0, \quad i \in \mathcal{I} \cap \mathcal{A}(x^*) \tag{18.25}$$

Note que se conhecêssemos o conjunto de restrições ativas para o ponto x^*, $\mathcal{A}(x^*)$, então a solução de (18.1) se reduziria a encontrar uma solução do sistema KKT dado por (18.22)–(18.25) que, por sua vez, consiste em se resolver um sistema de equações lineares. Este é o princípio utilizado pelos métodos de conjunto ativo.

18.4 O Método de Conjunto Ativo para Problemas Quadráticos Convexos

Se conhêcessemos $\mathcal{A}(x^*)$ a priori para o ponto ótimo x^*, poderíamos resolver (18.1) por meio dos métodos desenvolvidos para P_Q contendo apenas restrições de igualdade, conforme formulação (18.10).

Já encontramos um método de conjunto ativo, o Simplex. Simplex inicia com um chute para o conjunto ativo (variáveis básicas) e se o chute é incorreto então o Simplex iterativamente faz uso de multiplicadores de Lagrange e gradientes de forma a remover um elemento de $\mathcal{A}(x^*)$ e introduzir um novo elemento. O método de conjunto ativo para PQ (Programação Quadrática) difere do Simplex no sentido de que o primeiro não necessariamente vai de um vértice da região factível para outro. Alguns iterandos podem se encontrar em outros pontos da fronteira e até mesmo no interior. O método de conjunto ativo para PQ pode ser implementado de três formas:

1) Método Primal

2) Método Dual

3) Método Primal-dual

Aqui nos concentraremos no método primal, o qual inicia computando uma solução factível x_0 e segue gerando iterandos factíveis[2]. O método encontra um passo que transforma o iterando corrente no próximo iterando por meio da solução de um subproblema quadrático, no qual um subconjunto das restrições é imposto como igualdades. Tal subconjunto é referido como conjunto de trabalho e denotado por W_k na k-ésima iteração. W_k consiste de todas as restrições de igualdade $i \in \mathcal{E}$ juntamente com algumas, não necessariamente todas, as desigualdades ativas.

Dado um iterando x_k e um conjunto de trabalho W_k, primeiramente verificamos se x_k minimiza $q(x)$ no subespaço definido pelo conjunto de trabalho W_k. Em caso negativo, calculamos um passo p por meio da solução do subproblema quadrático tomando como restrições de igualdade todas as restrições do conjunto ativo.

Primeiramente, expressamos o subproblema em termos do passo p, definido por:

$$\begin{array}{rcl} p & = & x - x_k \\ g_k & = & Gx_k + d \end{array} \tag{18.26}$$

e através da substituição de x na função objetivo do problema (18.20) definimos:

$$q(x) = q(x_k + p) = \frac{1}{2} p^T G p + g_k^T p + c \tag{18.27}$$

onde $c = \frac{1}{2} x_k^T G x_k + d^T x_k$ é um termo constante. Uma vez que podemos remover a constante c da função objetivo, sem modificar a solução do problema, podemos escrever o subproblema quadrático a ser resolvido na k-ésima iteração como segue:

$$\begin{array}{lll} SP_k: & \text{Minimize} & \frac{1}{2} p^T G p + g_k^T p \\ & \text{Sujeito a:} & \\ & & a_i^T p = 0, \qquad i \in W_k \end{array} \tag{18.28}$$

Denotamos a solução de (18.28) por p_k. Note que para cada $i \in W_k$ o termo $a_i^T x$ não se altera à medida que movemos na direção p_k, uma vez que $a_i^T(x_k + p_k) = a_i^T x_k = b_i$. Portanto uma vez que as restrições em W_k são satisfeitas por x_k, elas também são satisfeitas no ponto $x_k + \alpha p_k$, qualquer que seja o valor α.

Suponha, por um momento, que a solução ótima p_k para (18.28) é não nula. Temos então que decidir quão longe seguiremos na direção dada por p_k. Se $x_k + p_k$ é factível com respeito a todas as restrições, então definimos $x_{k+1} = x_k + p_k$. Caso contrário, definimos:

$$x_{k+1} = x_k + \alpha_k p_k \tag{18.29}$$

onde α_k é o maior passo com valor entre $[0, 1)$ tal que todas as restrições são satisfeitas. Podemos definir uma fórmula explicita para α_k considerando apenas as restrições $i \notin W_k$, pois toda restrição $i \in W_k$ será satisfeita com certeza.

- Se $a_i^T p_k \geq 0$ para algum $i \notin W_k$, então para todo $\alpha_k \geq 0$ temos que $a_i^T(x_k + \alpha_k p_k) \geq a_i^T x_k \geq b_i$, portanto a restrição será satisfeita.

[2]Surge a questão de como encontrar um iterando inicial x_0 que seja factível para o problema (18.20). De que forma você encontraria x_0?

- Se $a_i^T p_k < 0$ para algum $i \notin W_k$, temos que $a_i^T(x_k + \alpha_k p_k) \geq b_i$ somente se:

$$\alpha_k \leq \frac{b_i - a_i^T x_k}{a_i^T p_k} \tag{18.30}$$

Uma vez que α_k deve ser o maior possível dentro da faixa $[0, 1)$ e tal que as restrições sejam respeitadas, obtemos a seguinte definição:

$$\alpha_k = \min\left\{1, \min\left\{\frac{b_i - a_i^T x_k}{a_i^T p_k} : i \notin W_k, a_i^T p_k < 0\right\}\right\} \tag{18.31}$$

A restrição i na qual o mínimo de (18.31) é obtido é chamada de *restrição bloqueante*. Note que é possível que α_k seja zero, uma vez que podemos ter uma restrição i que seja ativa para x_k mas não seja um membro de W_k.

Se $\alpha_k < 1$, ou seja, o passo ao longo de p_k foi bloqueado por alguma restrição que não pertence a W_k, então um novo conjunto de trabalho W_{k+1} é construído adicionando-se uma restrição bloqueante a W_k.

Continuamos este processo iterativo, adicionando restrições ao conjunto de trabalho, até que se atinja um ponto $\hat{x}$ que minimiza o objetivo quadrático sob o conjunto de trabalho $\hat{W}$. É fácil reconhecer tal ponto $\hat{x}$ pois o subproblema SP_k terá uma solução $p_k = 0$. Uma vez que $p_k = 0$ satisfaz as condições de otimalidade (18.13) do subproblema SP_k, temos que:

$$\begin{aligned} \sum_{i \in \hat{W}} a_i \hat{\lambda}_i &= \hat{g} \\ &= G\hat{x} + d \end{aligned} \tag{18.32}$$

para algum conjunto de multiplicadores de Lagrange $\hat{\lambda}_i, i \in \hat{W}$. Segue, portanto, que $\hat{x}$ e $\hat{\lambda}$ satisfazem as condições KKT (18.22), isso se definirmos os multiplicadores das restrições que não aparecem em $\hat{W}$ como zero.

Uma vez que o controle imposto no comprimento do passo garante a factibilidade de $\hat{x}$ com respeito a todas as restrições, temos que a segunda e a terceira condição KKT, (18.23) e (18.24) respectivamente, são também satisfeitas por $\hat{x}$.

Agora devemos examinar os sinais dos multiplicadores das restrições de desigualdade contidas no conjunto de trabalho $\hat{W}$, ou seja, os índices $i \in \hat{W} \cap \mathcal{I}$. Se todos estes multiplicadores são não-negativos, a quarta condição KKT (18.25) é satisfeita e $\hat{x}$ é um ponto KKT para o problema original. Se G é positiva semi-definida então $\hat{x}$ é também um mínimo local.

Por outro lado, se um multiplicador $\hat{\lambda}_j, j \in \hat{W} \cap \mathcal{I}$ é negativo, a condição (18.25) não é satisfeita, então a função objetivo $q(x)$ pode ser reduzida se removermos a restrição j de $\hat{W}$. Neste caso removeremos j de $\hat{W}$ para todo j tal que $\lambda_j < 0$ e resolvemos um novo subproblema (18.28).

O pseudo-código completo do algoritmo é dado na seqüência.

Algoritmo de Conjunto Ativo

Encontre um ponto factível x_0
Defina W_0 como um subconjunto das restrições ativas para x_0
Para $k = 0, 1, 2, \ldots$
    Resolva o subproblema SP_k
    Se $p_k = 0$
        Calcule os multiplicadores de Lagrange $\hat{\lambda}_i$ que satisfazem (18.32)
        Defina $\hat{W} = W_k$
        Se $\hat{\lambda}_i \geq 0$ para todo $i \in W_k \cap \mathcal{I}$
            Pare, a solução $x^* = x_k$ e ótima
        Caso contrário
            Defina $j = \text{argmin } \{\hat{\lambda}_j : j \in W_k \cap \mathcal{I}\}$
            $x_{k+1} \leftarrow x_k$
            $W_{k+1} \leftarrow W_k - \{j\}$
        Fim-Se
    Caso contrário ($p_k \neq 0$)
        Calcule α_k a partir de (18.31)
        $x_{k+1} \leftarrow x_k + \alpha_k p_k$
        Se há restrições bloqueantes
            Obtenha W_{k+1} adicionando uma restrição bloqueante a W_k
        Caso contrário
            $W_{k+1} \leftarrow W_k$
        Fim-Se
    Fim-Se
Fim-Para

18.4.1 Exemplo

Vamos aplicar o algoritmo de conjunto ativo ao problema bidimensional ilustrado na Figura 18.1 e definido por:

$$\begin{array}{ll} \text{Minimize} & q(x) = (x_1 - 1)^2 + (x_2 - 2.5)^2 \\ \text{Sujeito a :} & \\ & \begin{array}{lcl} x_1 - 2x_2 + 2 & \geq & 0 \\ -x_1 - 2x_2 + 6 & \geq & 0 \\ -x_1 + 2x_2 + 2 & \geq & 0 \\ x_1, x_2 & \geq & 0 \end{array} \end{array} \tag{18.33}$$

Numeramos as restrições de 1 até 5.

Iteração 0: Para esse problema é fácil de encontrar um ponto factível. Digamos que o ponto inicial é $x^0 = (2, 0)$, para o qual as restrições 3 e 5 estão ativas, portanto $W_0 = \{3, 5\}$. Note que poderíamos ter escolhido $W_0 = \{3\}$, $W_0 = \{5\}$ e até mesmo $W_0 = \emptyset$, mas cada uma dessas opções afetaria o comportamento do algoritmo.

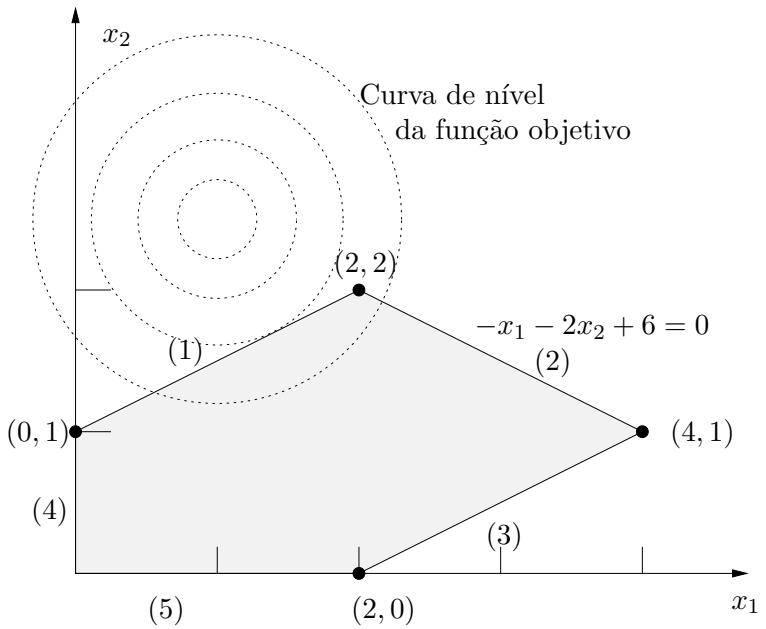


Figura 18.1: Região factível e curvas de nível da função objetivo.

Uma vez que x^0 corresponde a um vértice da região factível, ele é obviamente um minimizador da função objetivo com respeito ao conjunto de trabalho W_0, ou seja, a solução de (18.28) com $k = 0$ é $p_k = 0$ e, portanto, $x^1 = x^0$. Podemos então utilizar (18.32) para encontrar equações associadas com as restrições ativas. A partir da substituição dos dados de nosso problema em (18.32) obtemos:

$$\begin{bmatrix} -1 \\ 2 \end{bmatrix} \hat{\lambda}_3 + \begin{bmatrix} 0 \\ 1 \end{bmatrix} \hat{\lambda}_5 = \begin{bmatrix} 2 \\ -5 \end{bmatrix} \tag{18.34}$$

cuja solução é $(\hat{\lambda}_3, \hat{\lambda}_5) = (-2, -1)$

Podemos agora remover a restrição 3 do conjunto ativo uma vez que esta apresenta o multiplicador mais negativo, portanto, $W_1 = \{5\}$.

Iteração 1: Iniciamos a iteração 1 buscando uma solução de SP_k para $k = 1$, a qual é $p_1 = (-1, 0)$. A fórmula de comprimento de passo (18.31) produz $\alpha_1 = 1$, e um novo iterando $x^2 = (1, 0)$ é obtido.

Não há restrições bloqueantes, portanto $W_2 = W_1 = \{5\}$.

Iteração 2: Descobrimos no início da iteração 2 que a solução de (18.28) é novamente $p_2 = 0$, implicando em $x^3 = x^2 = (1, 0)$. A partir de (18.32) deduzimos que o multiplicador de Lagrange para a única restrição ativa é $\hat{\lambda}_5 = -5$. Assim, eliminamos a restrição 5 do conjunto de trabalho obtendo $W_3 = \emptyset$.

Iteração 3: A iteração 3 tem início com a solução do problema irrestrito, pois $W_3 = \emptyset$, para o qual obtemos a solução $p_3 = (0, 2.5)$. A fórmula (18.31) nos dá o comprimento do passo $\alpha_3 = 0.6$ e um novo iterando $x^4 = (1, 1.5)$. Há apenas uma restrição bloqueante: a restrição 1. Daí deduzimos que $W_4 = \{1\}$.

Iteração 4: A solução de (18.28) com $k = 4$ produz $p_4 = (0.4, 0.2)$, e o próximo passo é de comprimento 1. Não há restrições bloqueantes o que deixa o conjunto de trabalho na sua forma: $W_5 = \{1\}$. O novo iterando é $x^5 = (1.4, 1.7)$.

Iteração 5: Finalmente, resolvemos SP_k com $k = 5$ o que nos leva a obter a solução $p_5 = 0$. A fórmula (18.32) nos dá o multiplicador de Lagrange $\hat{\lambda}_1 = 1.25$. Portanto uma solução foi encontrada.

A solução ótima é $x^* = (1.4, 1.7)$. A Figura 18.2 ilustra a trajetória de iterandos, desde o ponto inicial x^0 até a solução ótima x^*.

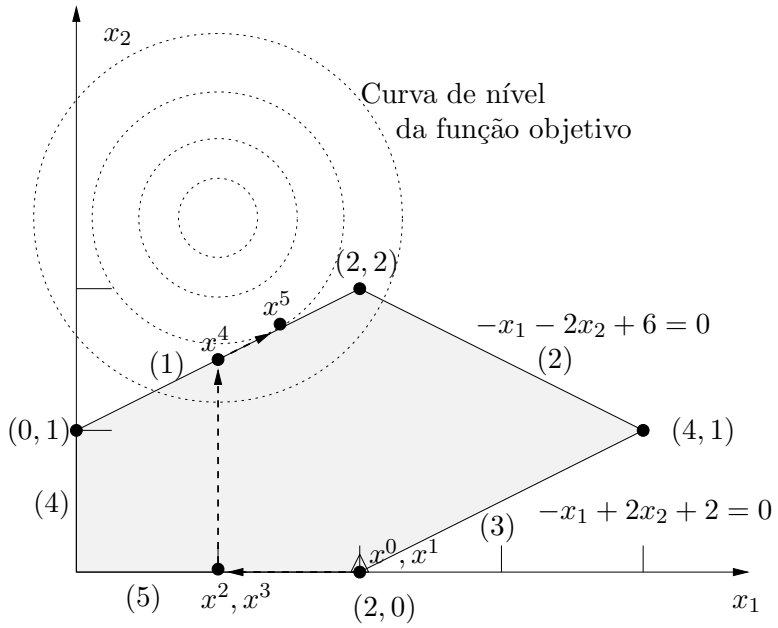


Figura 18.2: Seqüência de iterandos.

18.5 Exercícios

EX 18.1: Para o problema abaixo, P, execute as seguintes tarefas:

i) resolva o problema quadrático abaixo graficamente ou utilizando um pacote do site `http://www-neos.mcs.anl.gov/neos/server-solvers.html`;

ii) ilustre graficamente o espaço de soluções factíveis e a solução encontrada;

iii) verifique as condições de otimalidade de primeira ordem (as condições KKT); e

iv) aplique o algoritmo de conjunto ativo e verifique que a solução obtida confere com àquela obtida no item (i).

$$
\begin{array}{lll}
P: & \text{Maximize} & f(x) = 2x_1 + 3x_2 + 4{x_1}^2 + 2x_1x_2 + {x_2}^2 \\
& \text{Sujeito a :} & \\
& & \begin{array}{rcl} x_1 - x_2 & \geq & 0 \\ x_1 + x_2 & \leq & 4 \\ x_1 & \leq & 3 \end{array}
\end{array}
$$

EX 18.2: O problema de encontrar a menor distância de um ponto x_0 a um hiperplano $\{x : Ax = b\}$ onde A é uma matriz com posto de linha completo, $A \in \mathbb{R}^{m\times n}$ e $rank(A) = m$, portanto $n \geq m$, pode ser formulado como o problema quadrático abaixo:

$$
\begin{array}{ll}
\text{Minimize} & \frac{1}{2}(x - x_0)^T(x - x_0) \\
\text{Sujeito a :} & \\
& Ax = b
\end{array}
$$

Mostre que os multiplicadores de Lagrange ótimos são:

$$\lambda^* = (AA^T)^{-1}(b - Ax_0).$$

Usando λ^*, encontre a solução ótima x^*. (Sugestão: aplique as condições KKT.)

EX 18.3: Partindo do ponto inicial $x^0 = (3, 0)$ e com conjunto ativo $\mathcal{A}_0 = \{2\}$ aplique o algoritmo de conjunto ativo e encontre a solução global ótima do problema:

$$
\begin{array}{ll}
\text{Minimize} & \|(x_1, x_2)\|^2 \\
\text{Sujeito a :} & \\
& c_1(x) = x_1 + x_2 - 2 \geq 0 \\
& c_2(x) = 3 - x_1 - \frac{2}{3}x_2 \geq 0
\end{array}
$$

EX 18.4: Considere o problema de encontrar $x \in \mathbb{R}^n$ tal que:

$$
\begin{array}{ll}
\text{Minimize} & \|x\| \\
\text{Sujeito a :} & \\
& Ax = b
\end{array}
$$

onde $A \in \mathbb{R}^{m\times n}$. Prof. Kunz afirma que não faz sentido resolver este problema quando há mais equações do que variáveis, ou seja, quando $m > n$. Você concorda ou discorda? Justifique a resposta.

EX 18.5: Prof. Kunz desenvolveu um algoritmo para resolver problemas quadráticos dados por:

$$
\begin{array}{lll}
P_Q: & \text{Minimize} & q(x) = \frac{1}{2}x^TQx + d^Tx + c \\
& \text{Sujeito a :} & \\
& & Ax \geq b
\end{array}
$$

onde $Q \in \mathbb{R}^{n\times n}$ é uma matriz simétrica e positiva definida, $Q = Q^T$ e $Q > 0$. Suponha que você tem um problema idêntico a P_Q, exceto que a matriz $Q \neq Q^T$. Prof. Kunz afirma que seu algoritmo é capaz de resolver o problema em questão. Se você discorda do Prof. Kunz dê um contra-exemplo. Se você concorda, mostre como que o algortimo de Kunz pode ser utilizado.

EX 18.6: Considere o problema de programação quadrática:

$$P: \begin{array}{rl} \text{Minimize} & \frac{1}{2}w^T w + \frac{c}{2}\sum_{i=1}^{m} e_i^2 \\ w, b, e & \\ \text{Sujeito a :} & \\ & y_i = w^T\phi(x_i) + b + e_i,\ i = 1, \ldots, m \end{array}$$

onde:

- c é uma constante positiva;
- $\phi(x) : \mathbb{R}^n \to \mathbb{R}^p$ é um função dada, tipicamente uma função não-linear de $x \in \mathbb{R}^n$;
- $(x_i, y_i) \in \mathbb{R}^n \times \mathbb{R}, i = 1, \ldots, m$, são pares entrada-saída de uma função $f(x)$ desconhecida que estamos tentando aproximar por $\hat{f}(x, w, b) = w^T\phi(x) + b$;
- $e_i = y_i - \hat{f}(x_i, w, b) = y_i - (w^T\phi(x_i) + b)$ é o erro de aproximação para o i-ésimo par; e
- $w \in \mathbb{R}^p$ e $b \in \mathbb{R}$ são os parâmetros que definem a aproximação $\hat{f}$.

Fazendo uso do Lagrangeano de P, mostre que a solução ótima de P satisfaz a relação:

$$\begin{bmatrix} 0 & \mathbf{1}^T \\ \mathbf{1} & Q + c^{-1}I \end{bmatrix} \begin{bmatrix} b \\ \lambda \end{bmatrix} = \begin{bmatrix} 0 \\ y \end{bmatrix}$$

com $y = (y_1, \ldots, y_m)^T$, $\mathbf{1} = (1, \ldots, 1)^T$, $\lambda = (\lambda_1, \ldots, \lambda_m)^T$ e $Q_{ij} = \phi(x_i)^T\phi(x_j) = k(x_i, x_j)$ para $i, j = 1, \ldots, m$. Este tipo de aproximação é conhecido como *Support Vector Machine*. A função $k(x, x')$ é denominada de *kernel*. Dois tipos de *kernel* são:

i. polinômio de ordem q: $k(x, x') = (1 + x^T x')^q$

ii. *kernel* de base radial: $k(x, x') = \exp(\frac{\|x - x'\|_2^2}{2\sigma^2})$

EX 18.7: Considere a matrix $(n + 1) \times (n + 1)$ conhecida por *matrix de borda* dada por:

$$G = \begin{bmatrix} A & u \\ u^T & a \end{bmatrix} \tag{18.35}$$

onde A é uma matriz $n \times n$, $u \in \mathbb{R}^{n\times 1}$ e $a \in \mathbb{R}$. Conhecendo A^{-1}, desejamos encontrar uma forma rápida de calcular G^{-1}. Seja a inversa de G estruturada como:

$$G^{-1} = \begin{bmatrix} B & q \\ q^T & b \end{bmatrix} \tag{18.36}$$

Fazendo $B = A^{-1} + bA^{-1}uu^TA^{-1}$ e $b = 1/(a - u^TA^{-1}u)$, encontre q tal que (18.36) defina a inversa de G.

Capítulo 19

Programação Não-Linear sob Restrições: Algoritmos

Os capítulos anteriores apresentaram fundamentos de otimização não-linear sob restrições, estabelecendo condições de otimalidade local de primeira ordem e desenvolvendo algoritmos para resolver problemas quadráticos. Este capítulo tem foco em algoritmos para resolver problemas gerais desta classe de problemas.

19.1 Algoritmos

Aqui nos concentraremos em algoritmos para resolver (encontrar soluções ótimas locais) o problema geral de programação não-linear:

$$\begin{array}{rll} \text{Minimize} & f(x) & \\ x \in \Re^n & & \\ \text{Sujeito a :} & & \\ & c_i(x) = 0, \quad & i \in \mathcal{E} \\ & c_i(x) \geq 0, \quad & i \in \mathcal{I} \end{array} \tag{19.1}$$

onde as funções são suaves. Uma classe importante de algoritmos, a serem vistos a seguir, busca uma solução para (19.1) através da solução de uma seqüência de subproblemas irrestritos.

19.2 O Método de Barreira Logarítmica

Os métodos de barreira logarítmica resolvem uma série de problemas irrestritos, aproximando o problema original, que no limite se tornam equivalentes ao problema original. O método de barreira usa uma função de penalidade (barreira) que cresce rapidamente à medida que o ponto corrente se aproxima da fronteira da região de factibilidade, dessa forma garantindo factibilidade da solução corrente. A cada iteração, a penalidade é reduzida para permitir que a solução ótima seja alcançada.

19.2.1 Propriedades das Funções de Barreira Logarítmica

Iniciamos a discussão por meio de uma ilustração das funções de barreira para problemas com apenas restrições de desigualdade.

$$\begin{array}{ll} \text{Minimize} & f(x) \\ x & \\ \text{Sujeito a :} & c_i(x) \geqslant 0, \quad i \in \mathcal{I} \end{array} \tag{19.2}$$

A região de factibilidade estrita é definida por

$$\mathcal{F}^0 = \{x \in \mathbb{R}^n : c_i(x) > 0, \quad \text{para todo } i \in \mathcal{I}\} \tag{19.3}$$

Assumimos que $\mathcal{F}^0$ é não-vazia para o propósito desta discussão. Funções barreira para este problema têm as seguintes propriedades:

a) elas são infinitas em qualquer lugar, menos em $\mathcal{F}^0$.

b) elas são suaves dentro de $\mathcal{F}^0$.

c) O valor delas se aproxima de ∞ à medida que x se aproxima da fronteira de $\mathcal{F}^0$.

A função barreira mais importante é a *função barreira logarítmica*, que para o conjunto de restrições $c_i(x) \geqslant 0$, $i \in \mathcal{I}$, assume a forma:

$$-\sum_{i \in \mathcal{I}} \log c_i(x) \tag{19.4}$$

onde $\log(y)$ denota o logaritmo natural de y. Para o problema (19.2), a função combinada objetivo-barreira é dada por:

$$P(x, \mu) = f(x) - \mu \sum_{i \in \mathcal{I}} \log c_i(x) \tag{19.5}$$

onde $\mu \geqslant 0$ é dito parâmetro barreira. Daqui em diante, vamos nos referir a $P(x, \mu)$ como *função barreira logarítmica do problema (19.2).*

Exemplo 1

Considere o problema abaixo em uma variável.

$$\begin{array}{lrcl} \text{Minimize} & x & & \\ \text{Sujeito a :} & x & \geqslant & 0 \\ & 1 - x & \geqslant & 0 \end{array} \tag{19.6}$$

Para o qual temos:

$$P(x, \mu) = x - \mu \log x - \mu \log (1 - x) \tag{19.7}$$

A Figura 19.1 ilustra $P(x, \mu)$ para $\mu = 1.0, 0.4, 0.1, 0.01$. Naturalmente, para valores pequenos de μ, a função $P(x, \mu)$ se aproxima da função objetivo f na maior parte da região factível; a função $P(x, \mu)$ tende para $+\infty$ nas proximidades da fronteira.

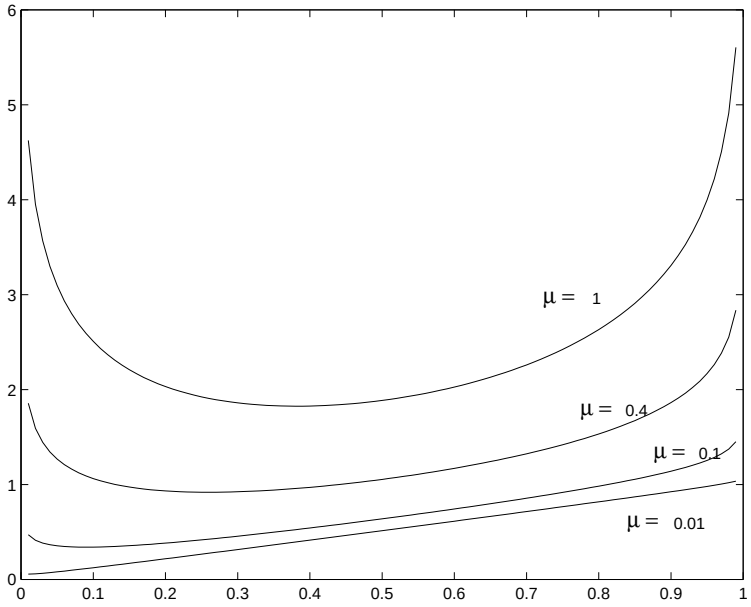


Figura 19.1: Curvas de nível da função $P(x,\mu) = x - \mu\log x - \mu\log(1-x)$ para $\mu \in \{0.01, 0.1, 0.4, 1\}$

Na Figura 19.1, a curva $P(x, 0.01)$ é quase indistinguível de $f(x)$ para a resolução da figura, mas a função tende para $+\infty$ nos extremos.

É evidente que quando $\mu \to 0$, o minimizador $x(\mu)$ de $P(x,\mu)$ se aproxima da solução $x^* = 0$ do problema (19.6). Uma vez que o minimizador $x(\mu)$ de $P(x,\mu)$ está dentro de $\mathcal{F}^0$, podemos em princípio executar uma busca utilizando algoritmos de minimização irrestritos. Infelizmente, o minimizador $x(\mu)$ torna-se progressivamente mais difícil de ser encontrado à medida que $\mu \to 0$.

Exemplo 2

Considere o problema:

$$\begin{array}{ll} \text{Minimize} & \left(x_1 + \frac{1}{2}\right)^2 + \left(x_2 - \frac{1}{2}\right)^2 \\ \text{Sujeito a :} & \\ & x_1 \in [0,1] \\ & x_2 \in [0,1] \end{array} \tag{19.8}$$

Para o qual a função barreira é:

$$\begin{array}{rcl} P(x,\mu) & = & (x_1 + \frac{1}{2})^2 + \left(x_2 - \frac{1}{2}\right)^2 \\ & & -\mu\left[\log x_1 + \log(1-x_1) + \log x_2 + \log(1-x_2)\right] \end{array} \tag{19.9}$$

As curvas de nível da função (19.9) são ilustradas na Figura 19.2 para os valores $\mu = 22$, $\mu = 5$ e $\mu = 2$. A partir da figura, observamos que as curvas de nível se aproximam da curva de uma parábola à medida que $\mu \to 0$. As curvas de nível que envolvem o minimizador, $(x_1^* = 0, x_2^* = \frac{1}{2})$, nos casos onde $\mu = 22$ e $\mu = 5$ não são muito alongadas, sendo quase elípticas. Neste caso podemos aplicar métodos de minimização

irrestrita sem maiores problemas. Para o valor $\mu = 2$, entretanto, as curvas de nível são mais alongadas e menos elípticas quando o minimizador $x(\mu)$ é deslocado em direção à fronteira da função barreira. O caráter alongado das curvas de nível indica normalização pobre, acarretando baixo desempenho de métodos de otimização irrestritos, como por exemplo, o algoritmo de descenso.

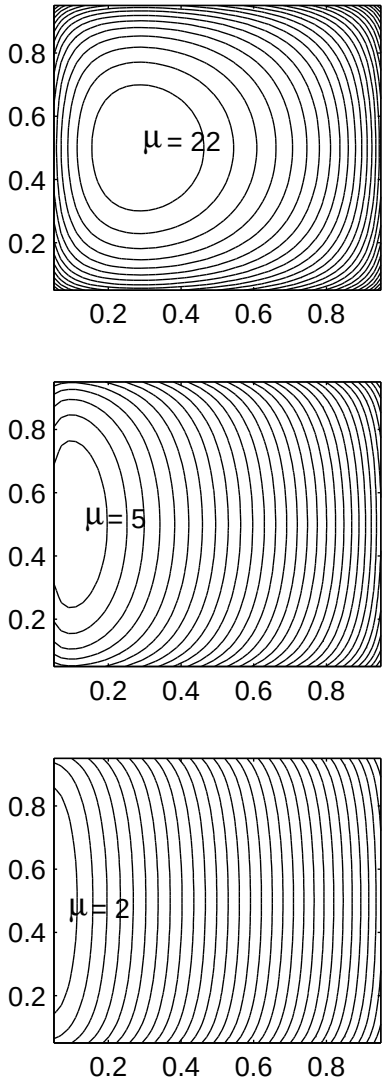


Figura 19.2: Curvas de nível da função $P(x, \mu) = (x_1 + \frac{1}{2})^2 + (x_2 - \frac{1}{2})^2 - \mu\left(\log x_1 + \log(1 - x_1) + \log x_2 + \log(1 - x_2)\right)$ para $\mu \in \{22, 5, 2\}$

19.2.2 Algoritmo Baseado na Função Barreira Logarítmica

Os algoritmos baseados na função barreira logarítmica procuram aproximar os minimizadores de $P(x, \mu)$ para uma seqüência decrescente de valores μ.

`Algoritmo Log-Barrier (Esboço)`

Dados $\mu_o > 0$, tolerância $\tau > 0$ e um ponto inicial x_0^S
Para $k = 0, 1, 2, ...$ faça
▷ Encontre um minimizador aproximado x_k para $P(., \mu_k)$
tomando como ponto de partida x_k^S e
terminando com $\|\nabla P(x_k, \mu_k)\| \leqslant \tau$.
▷ Se o critério final de convergência foi satisfeito então pare,
retorne a solução aproximada x_k.
▷ Escolha um novo parâmetro barreira $\mu_{k+1} \epsilon (0, \mu_k)$.
▷ Escolha um novo ponto inicial fazendo $x_{k+1}^S = x_k$.
Fim-Para

Para garantir que o método de Newton convirja rapidamente para o minimizador $x(\mu_k)$, as seguintes técnicas podem ser utilizadas:

a) Utilize técnicas que garantem convergência global, como busca direcional e mecanismos de região de confiança. Essas técnicas devem garantir que os iterandos permaneçam dentro da região factível $\mathcal{F}^0$.

b) Um ponto inicial x_k^S promissor para $P(x, \mu_k)$ pode ser obtido extrapolando-se ao longo do caminho definido pelos minimizadores $x_{k-1}, x_{k-2}, \ldots$.

Várias heurísticas foram propostas para a escolha do parâmetro de barreira μ_{k+1} a cada iteração. Um ingrediente típico é fazer escolhas ambiciosas para μ_{k+1} (digamos $\mu_{k+1} = 0.2\mu_k$ ou $\mu_{k+1} = 0.1\mu_k$) se o problema de minimização $P(x, \mu_k)$ não foi tão difícil de ser resolvido na última iteração e se um ponto inicial x_{k+1}^S pode ser produzido com certa confiança.

O algoritmo básico descrito acima ainda não se tornou um algoritmo de uso geral, não permitindo que uma resposta definitiva sobre a qualidade das heurísticas seja dada sem testes extensivos.

19.2.3 Propriedades da Função Log-Barrier e Algoritmo Geral

Teorema 19.1 *Suponha que f e $-c_i$, $i \in \mathcal{I}$, sejam convexas em (19.2) e a região factível $\mathcal{F}^0$ seja não-vazia. Seja $\{\mu_k\}$ uma sequência decrescente, e assuma que o conjunto de soluções M para o problema (19.2) seja não-vazio e limitado. Então as seguintes afirmações são verdadeiras:*

i) Para qualquer $\mu > 0$, $P(x, \mu)$ é convexo em $\mathcal{F}^0$ e admite um minimizador $x(\mu)$ (não necessariamente único) em $\mathcal{F}^0$. Qualquer minimizador $x(\mu)$ é também um minimizador global para $P(x, \mu)$.

ii) Qualquer seqüência de minimizadores $x(\mu_k)$ tem uma subseqüência convergente, e todos os possíveis pontos de acumulação pertencem a M.

iii) $f(x(\mu_k)) \to f^*$ e $P(x(\mu_k), \mu_k) \to f^*$ para qualquer seqüência de minimizadores $\{x(\mu_k)\}$.

Para problemas mais gerais, o resultado correspondente é mais local por natureza. Dada uma solução local x^* "bem comportada" do problema (19.2), isto é, uma solução x^* que satisfaça as condições suficientes de segunda-ordem, as restrições e ainda as condições de complementaridade, a função barreira $P(x, \mu)$ apresenta um minimizador próximo de x^* para valores suficientemente pequenos de μ.

Teorema 19.2 *Suponha que $\mathcal{F}^0$ é uma região não-vazia e que x^* é uma solução para o problema (19.2), o qual satisfaz as condições KKT (Karush-Kuhn-Tucker) para algum λ^*. Suponha também que a condição LCG de qualificação de restrições seja garantida, bem como a condição de complementaridade estrita e as condições suficientes de segunda ordem.*

Então, as afirmações abaixo são verdadeiras:

i) Existe uma função vetorial $x(\mu)$ contínua, única e diferenciável, definida para todo μ suficientemente pequeno em função de que $x(\mu)$ é um minimizador local de $P(x, \mu)$ em uma certa vizinhança de x^k, tal que $\lim_{\mu \to 0} x(\mu) = x^*$.

ii) Para a função $x(\mu)$ em (i), o estimador $\lambda(\mu)$ dos multiplicadores de Lagrange definidos por

$$\lambda_i(\mu) \equiv \frac{\mu}{c_i(x(\mu))}, i \in \mathcal{I} \tag{19.10}$$

convergem para λ^* à medida que $\mu \to 0$.

iii) A matriz Hessiana $\nabla^2_{xx} P(x, \mu)$ é positiva definida para todo μ suficientemente pequeno.

A trajetória c_p definida por

$$c_p \equiv \{x(\mu) : \mu > 0\} \tag{19.11}$$

é frequentemente referenciada como *Caminho Central* (ou Caminho Central Primal).

19.2.4 Manipulando Restrições de Igualdade

Até este momento assumimos que o problema apresenta apenas restrições de desigualdade e que $\mathcal{F}^0$ é não-vazio. Aqui discutimos um procedimento que nos permite aplicar a técnica de Barreira Logarítmica ao problema geral 19.1.

Não podemos simplesmente reescrever $c_i(x) = 0$ como duas desigualdades $c_i(x) \geqslant 0$ e $-c_i(x) \geqslant 0$, já que isto tornaria $\mathcal{F}^0$ vazio. Uma forma de tratar as restrições de igualdade é incluir termos quadráticos de penalidade para estas restrições na função objetivo. Se assumimos, por um momento, que o coeficiente da penalidade quadrática é $\frac{1}{\mu}$, onde μ é o parâmetro barreira, então a função *barreira logarítmica/penalidade quadrática* a ser minimizada para cada valor μ é:

$$B(x, \mu) \equiv f(x) - \mu \sum_{i \in \mathcal{I}} \log c_i(x) + \frac{1}{2\mu} \sum_{i \in \mathcal{E}} c_i^2(x) \tag{19.12}$$

Como esperado, as propriedades desta função combinam as propriedades da *função barreira logarítmica* e da *função penalidade quadrática.*

O algoritmo geral apresentado na seção 19.2.2 pode ser estendido de forma que reduções sucessivas de μ e minimizações aproximadas de $B(x, \mu)$ nos levam a uma solução para 19.1.

19.3 Método Lagrangeano Aumentado

Este método tende a produzir sub-problemas menos mal-condicionados do que aqueles gerados pelo método de barreira logarítmica. As implementações podem ser construídas a partir de software padrão para otimização irrestrita. O método de multiplicadores é a base de implementações práticas de alta qualidade, como LANCELOT por exemplo.

19.3.1 Motivação e Estrutura do Algoritmo

Consideramos inicialmente o problema com restrições de igualdade:

$$\begin{array}{ll} \text{Minimize} & f(x) \\ \text{Sujeito a :} & c_i(x) = 0, \quad i \in \mathcal{E} \end{array} \tag{19.13}$$

A função penalidade quadrática $Q(x; \mu)$ penaliza as violações das restrições elevando ao quadrado as infactibilidades e normalizando-as com o fator $\frac{1}{\mu}$. Mas a penalidade $Q(x; \mu)$ apresenta problemas de condicionamento. Nesse sentido, a função Lagrangeana aumentada busca evitar problemas de condicionamento ao introduzir explicitamente os estimadores dos multiplicadores de Lagrange à função penalidade quadrática. A função Lagrangeana aumentada é definida como:

$$\mathcal{L}_A(x, \lambda; \mu) \stackrel{def}{=} f(x) - \sum_{i \in \mathcal{E}} \lambda_i c_i(x) + \frac{1}{2\mu} \sum_{i \in \mathcal{E}} c_i^2(x) \tag{19.14}$$

Primeiramente, vamos nos lembrar da função Lagrangeana padrão e da função de penalidade quadrática.

$$\mathcal{L}(x, \lambda) = f(x) - \sum_{i \in \mathcal{E}} \lambda_i c_i(x) \tag{19.15a}$$

$$\mathcal{Q}(x; \mu) = f(x) + \frac{1}{2\mu} \sum_{i \in \mathcal{E}} c_i^2(x) \tag{19.15b}$$

Observa-se que a função Lagrangeana aumentada (19.14) difere da função Lagrangeana padrão devido à presença da penalidade quadrática, e também difere da função penalidade quadrática (19.15a) devido à presença do termo com os multiplicadores de Lagrange. Assim, $\mathcal{L}_A$ é uma combinação de (19.15a) e (19.15b).

Diferenciando-se $\mathcal{L}_A$ com respeito à x, obtém-se:

$$\nabla_x \mathcal{L}_A(x, \lambda; \mu) = \nabla f(x) - \sum_{i \in \mathcal{E}} \left[\lambda_i - \frac{c_i(x)}{\mu} \right] \nabla c_i(x) \tag{19.16}$$

A seguir, vamos desenvolver um algoritmo que fixa o parâmetro barreira μ a um certo valor $\mu^k > 0$ na k-ésima iteração, semelhante ao algoritmo apresentado na seção (19.2.2), fixa λ na estimativa corrente λ^k, e executa minimização com respeito a x. Usando x^k para denotar o minimizador aproximado de $\mathcal{L}_A(x, \lambda; \mu)$ e fazendo uso de certos resultados teóricos, podemos deduzir que

$$\lambda_i^* \approx \lambda_i^k - \frac{c_i(x^k)}{\mu^k} \text{ para todo } i \in \mathcal{E} \tag{19.17}$$

Rearranjando (19.17), obtemos:

$$c_i(x^k) \approx -\mu^k \left(\lambda_i^* - \lambda_i^k\right) \text{ para todo } i \in \mathcal{E} \tag{19.18}$$

Portanto, podemos concluir que se λ^k é próximo do multiplicador ótimo λ^*, então a infactibilidade em x^k será muito menor do que μ^k, não sendo mais proporcional a μ^k.

Como poderíamos atualizar a estimativa λ^k de uma iteração para a seguinte, de forma que a estimativa gradualmente se aproxime de λ^*, tendo como base a informação corrente? A equação (19.17) sugere a fórmula:

$$\lambda_i^{k+1} = \lambda_i^k - \frac{c_i(x^k)}{\mu^k} \text{ para todo } i \in \mathcal{E} \tag{19.19}$$

A discussão acima nos leva a um algoritmo geral:

`Algoritmo Lagrangeano Aumentado`

`Dados` $\mu^0 > 0$, `tolerância` $\tau^0 > 0$, `ponto inicial` x_s^0 `e` λ^0
`Para` $k = 0, 1, 2, \ldots$
- ▷ `Encontre um minimizador aproximado` x^k `para` $\mathcal{L}_A(., \lambda^k; \mu^k)$, `partindo de` x_s^k
- ▷ `Se o teste de convergência final é satisfeito, então pare e retorne a solução aproximada` x^k
- ▷ `Atualize os multiplicadores de Lagrange utilizando a fórmula` $\lambda_i^{k+1} = \lambda_i^k - \frac{c_i(x^k)}{\mu^k}$ `para` $i \in \mathcal{E}$
- ▷ `Escolha novas penalidades` $\mu^{k+1} \in (0, \mu^k)$
- ▷ `Defina o ponto inicial para a próxima iteração como` $x_s^{k+1} = x^k$

19.3.2 Exemplo

Considere o problema abaixo:

$$\begin{aligned} &\text{Minimize} && x_1 + x_2 \\ &\text{Sujeito a :} && \\ & && x_1^2 + x_2^2 - 2 = 0 \end{aligned} \tag{19.20}$$

Para o problema (19.20), a função Lagrangeana Aumentada correspondente é:

$$\mathcal{L}_A(x, \lambda; \mu) = x_1 + x_2 - \lambda(x_1^2 + x_2^2 - 2) + \frac{1}{2\mu}(x_1^2 + x_2^2 - 2)^2 \tag{19.21}$$

Sabemos que a solução ótima $x^* = (-1, -1)$ e que o multiplicador de Lagrange ótimo é $\lambda^* = \frac{1}{2}$. Suponha que na iteração k, temos $\mu^k = 1$ enquanto que a estimativa do multiplicador de Lagrange é $\lambda^k = -\frac{2}{5}$. O Minimizador de $\mathcal{L}_A(x, -0.4; 1)$ é aproximadamente $x^k = (-1.02; -1.02)$, que é mais próximo de x^* do que o minimizador de $Q(x; 1)$, que é aproximadamente $(-1.1, -1.1)$.

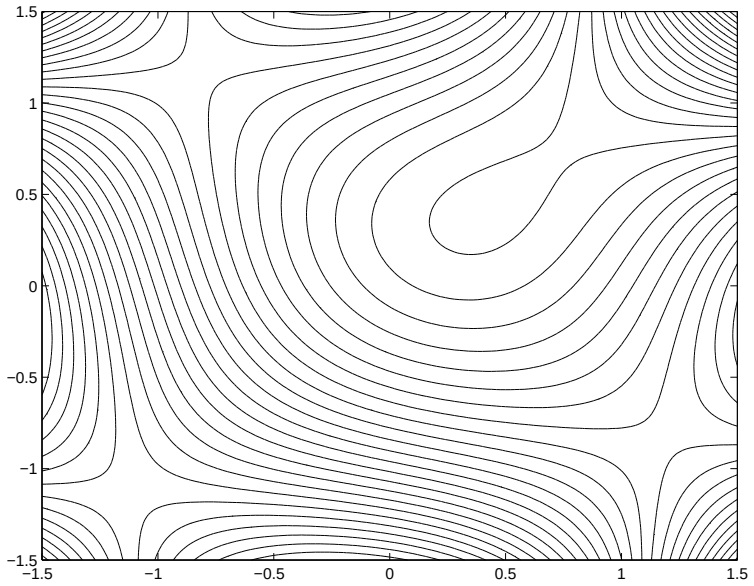


Figura 19.3: Curvas de nível da função $\mathcal{L}_A(x, -0.4, 50)$

19.3.3 Extensão para Restrições de Desigualdade

O primeiro passo para tratar o problema geral é substituir as desigualdades $c_i(x) \geqslant 0$, $i \in \mathcal{I}$, por

$$c_i(x) - s_i = 0, \;\; s_i \geqslant 0 \;\; \text{para todo } i \in \mathcal{I} \tag{19.22}$$

Podemos tratar das restrições $s_i \geqslant 0$, $i \in \mathcal{I}$, diretamente no subproblema Lagrangeano de maneira a eliminarmos as variáveis de folga s_i, $i \in \mathcal{I}$. Por razões de simplicidade, vamos assumir que não há restrições de igualdade ($\mathcal{E} = \emptyset$). A introdução das variáveis de folga produz o seguinte problema:

$$\begin{array}{ll} \text{Minimize} & f(x) \\ x, s & \\ \text{Sujeito a :} & c_i(x) - s_i = 0, \;\; s_i \geqslant 0, \;\; i \in \mathcal{I} \end{array} \tag{19.23}$$

Ao estabelecermos a função Lagrangeana aumentada em termos das restrições $c_i(x) - s_i = 0$ e aplicando as restrições $s_i \geqslant 0$ explicitamente, obtemos o seguinte subproblema a ser resolvido na iteração k do algoritmo apresentado na seção 19.3.1:

$$\begin{array}{ll} \text{Minimize} & \mathcal{L}_A(x, \lambda; \mu) = f(x) - \sum\limits_{i \in \mathcal{I}} \lambda_i (c_i(x) - s_i) + \frac{1}{2\mu} \sum\limits_{i \in \mathcal{I}} (c_i(x) - s_i)^2 \\ x, s & \\ \text{Sujeito a :} & s_i \geqslant 0 \;\; \text{para todo } i \in \mathcal{I} \end{array} \tag{19.24}$$

Note que cada s_i ocorre em dois termos de (19.24), que de fato é uma função convexa com respeito às variáveis de folga. Podemos portanto executar uma minimização explícita em (19.24) com respeito a cada s_i. A derivada parcial da função objetivo de (19.24) com respeito a s_i é:

$$\lambda_i^k - \frac{1}{\mu_k}(c_i(x) - s_i) \tag{19.25}$$

O minimizador irrestrito de (19.24) com respeito a s_i ocorre quando as derivadas parciais são nulas, ou seja,

$$s_i = c_i(x) - \mu_k \lambda_i^k \tag{19.26}$$

Se o minimizador irrestrito é menor do que 0, então porque (19.24) é convexa, o valor ótimo de s_i é 0. Resumindo, obtemos os valores ótimos para s_i:

$$s_i = \max\left\{c_i(x) - \mu\lambda_i^k, 0\right\} \text{ para todo } i \in \mathcal{I} \tag{19.27}$$

Podemos utilizar a fórmula (19.27) para eliminarmos s do subproblema (19.24). Isolando-se s_i em (19.24), temos que:

$$-\lambda_i^k(c_i(x) - s_i) + \frac{1}{2\mu}(c_i(x) - s_i)^2 = \begin{cases} -\lambda_i^k c_i(x) + \frac{1}{2\mu}c_i^2(x), & \text{se } c_i(x) - \mu\lambda_i^k \leqslant 0 \\ -\frac{\mu}{2}(\lambda_i^k)^2, & \text{caso contrário} \end{cases}$$

Definimos a função $\psi(t, \sigma; \mu)$ cujos argumentos são escalares, de maneira a representar a lógica de (**??**)

$$\psi(t, \sigma; \mu) = \begin{cases} -\sigma t + \frac{1}{2\mu}t^2, & \text{se } t - \mu\sigma \leqslant 0 \\ -\frac{\mu}{2}\sigma^2, & \text{caso contrário} \end{cases} \tag{19.28}$$

e utilizando-se a função ψ, pode-se expressar o subproblema transformado como:

$$\begin{array}{ll} \text{Min} & \mathcal{L}_A(x, \lambda; \mu) = f(x) + \sum_{i \in \mathcal{I}} \psi(c_i(x), \lambda_i^k; \mu^k) \\ x & \end{array} \tag{19.29}$$

A definição (19.29) de $\mathcal{L}_A$ é uma extensão natural do Lagrangeano aumentado para caso de desigualdades. Com esta extensão podemos aplicar o algoritmo delineado na Seção (19.3.1) para o caso de desigualdades, mas com uma pequena modificação: tão logo uma solução aproximada x^k de (19.29) seja obtida, utilizamos a fórmula abaixo para atualizar os multiplicadores de Lagrange:

$$\lambda_i^{k+1} = \max\left\{\lambda_i^k - \frac{c_i(x^k)}{\mu^k}, 0\right\} \quad \text{para todo } i \in \mathcal{I} \tag{19.30}$$

Observações: Cada função $\psi(c_i(x), \lambda_i; \mu)$ é continuamente diferenciável com respeito a x, mas há em geral uma descontinuidade na segunda derivada com respeito a x sempre que $c_i(x) = \mu\lambda_i$ para algum $i \in \mathcal{I}$. Entretanto, quando a condição de complementaridade é satisfeita (se apenas um dos termos λ_i^* e $c_i(x^*)$ é nulo), o minimizador x^k de cada subproblema (19.29) usualmente não é próximo da região de descontinuidade.

19.3.4 Propriedades do Lagrangeano Aumentado

Dois resultados encontrados na literatura justificam o uso do Lagrangeano aumentado e dos algoritmos desenvolvidos.

- Quando o valor exato dos multiplicadores de Lagrange λ^* são conhecidos, então a solução ótima x^* do problema (19.13) é também um minimizador estrito de $\mathcal{L}_A(x, \lambda^*; u)$ para u suficientemente pequeno.
- Embora não conheçamos λ^* na prática, os resultados mostram que podemos obter uma solução x muito próxima de x^* ao minimizarmos $\mathcal{L}_A(x, \lambda; u)$, mesmo quando u não está particularmente próximo de zero, desde que λ seja uma "boa" estimativa de λ^*.

19.4 Programação Linear Sequencial

A idéia principal por trás da programação linear sequencial (PLS) está na geração de um passo por meio da minimização do Lagrangeano sujeito a uma linearização das restrições.

Aqui nos concentraremos no problema de optimização não-linear sob restrições de igualdade, por razões de conveniência. Para o problema (19.13), o subproblema PLS assume a forma:

$$\begin{array}{ll} \text{Minimize} & F_k(x) \\ x & \\ \text{Sujeito a :} & \nabla c_i(x^k)^T(x - x^k) + c_i(x^k) = 0 \quad \forall i \in \mathcal{E} \end{array} \tag{19.31}$$

Existem diferentes escolhas para $F_k(x)$. Os primeiros métodos definiam:

$$F_k(x) = f(x) - \sum_{i \in \mathcal{E}} \lambda_i^k \hat{c}_i^k(x) \tag{19.32}$$

onde:

- λ_i^k é a estimativa corrente de λ_i^*; e
- $\hat{c}_i^k(x)$ é a diferença entre $c_i(x)$ e sua linearização no ponto x_k, ou seja,

$$c_i^{-k}(x) = C_i(x) - c_i(x^k) - \nabla c_i(x^k)^T(x - x^k) \tag{19.33}$$

É possível mostrar que à medida que x^k se aproxima de x^*, os multiplicadores de Lagrange associados com (19.31) convergem para os multiplicadores ótimos. Portanto, tomamos os estimadores de λ^k dados em (19.33) como os multiplicadores do subproblema (19.31) na iteração anterior.

Com o intuito de se obter convergência a partir de pontos iniciais remotos, os métodos PLS mais populares utilizam o Lagrangeano aumentado com F_k :

$$F_k(x) = f(x) - \sum_{i \in \mathcal{E}} \lambda_i^k \hat{c}_i^k(x) + \frac{1}{2\mu} \sum_{i \in \mathcal{E}} [\hat{c}_i^k(x)]^2 \tag{19.34}$$

onde μ é um parâmetro positivo de penalidade.

Métodos PLS são utilizados em problemas de larga-escala, sendo o modelo (19.34) incorporado na função objetivo em implementações bem-sucedidas como MINOS. Uma vez que as computações a cada passo exigem a minimização do subproblema não-linear (19.31), várias iterações se fazem necessárias e portanto, várias avaliações de funções e restrições para gerar um novo iterando. Mesmo assim, o número de iterações é menor do que em outros métodos.

O ponto fraco dos métodos PLS é a exigência de resolver os subproblemas com grande exatidão, para garantir que os multiplicadores de Lagrange sejam de "boa qualidade". Apesar da programação quadrática sequencial ser considerada superior, algoritmos clássicos como Minos que implementam o método PLS estão entre os melhores algoritmos.

19.5 Programação Quadrática Sequencial

Um dos métodos mais eficientes para resolver problemas de optimização não-linear sob restriçoẽs y programação quadrática sequencial (SQP, *"sequential quadratic programming"*), produz passos para o iterando corrente por meio da solução de problemas quadráticos. Diferentemente de programação linear sequencial, que é mais eficaz quando as restriçoẽs são lineares, SQP é robusto em problemas com não-linearidades significativas nas restrições.

19.5.1 O método SQP Local

Vamos iniciar a discussão considerando o problema não-linear com restriço es de igualdade:

$$\begin{array}{ll} Minimize & f(x) \\ Sujeito\ \ a: & \\ & C(x) = 0 \end{array} \tag{19.35}$$

Onde $f : \mathbb{R}^n \to \mathbb{R}$ e $c : \mathbb{R}^n \to \mathbb{R}^m$ são funçoẽs suaves. Problemas que contêm apenas restriçoẽs de igualdade não são muito comuns na prática, mas um entendimento de 19.35é crucial para o desenvolvimento de algoritmos para o caso geral.

A ideia de SQP advém do uso de um modelo quadrático de aproximação de 19.35 em torno do iterando x_k. A solução deste subproblema nos dá o próximo iterando, x_{k+1}.

O desafio está em se projetar o subproblema quadrático de forma que se obtenha um "bom"passo para o problema 19.35, dessa forma induzindo convergência rápida e robusta para um mínimo local.

Talvez a forma mais simple de se obter o método SQP é derivá-lo da aplicação do método de Newton para as condições KKT.

A partir da aplicação das condições KKT, sabemos que o Lagrangeano de 19.35 é dado por:

$$L(x, \lambda) = f(x) - \lambda^T C(x) \tag{19.36}$$

Utilizaremos $A(x)$ para denotar a matriz Jacobiana das restrições, ou seja:

$$A(x)^T = [\nabla C_1(x), ..., \nabla C_m(x)] \tag{19.37}$$

onde $C_i(x)$ é o i-ésimo componente de C(x). Especializando-se as condições KKT para o caso de igualdades, obtemos um sistema com $n+m$ equações e $n+m$ variáveis x e λ:

$$F(x,\lambda) = \begin{bmatrix} \nabla f(x) - A(x)^T\lambda \\ C(x) \end{bmatrix} = 0 \tag{19.38}$$

Um algoritmo consiste em se utilizar o método de Newton para o sistema não-linear 19.38. A matriz Jacobiana de 19.38 é dada por:

$$\begin{bmatrix} W(x,\lambda) & -A(x)^T \\ A(x) & 0 \end{bmatrix} \tag{19.39}$$

onde W denota a matriz Hessiana do Lagrangeano:

$$W(x,\lambda) = \nabla^2_{xx} L(x,\lambda) \tag{19.40}$$

O passo de Newton a partir do iterando (x_k, λ_k) é obtido como segue:

$$\begin{bmatrix} x_{k+1} \\ \lambda_{k+1} \end{bmatrix} = \begin{bmatrix} x_k \\ \lambda_k \end{bmatrix} + \begin{bmatrix} P_k \\ P_\lambda \end{bmatrix} \tag{19.41}$$

onde $P_k e P_\lambda$ resolvem o sistema KKT abaixo:

$$\begin{bmatrix} W_k & -A_k^T \\ A_k & \end{bmatrix} \begin{bmatrix} P_k \\ P_\lambda \end{bmatrix} = \begin{bmatrix} -\nabla f_k + A_k^T\lambda_k \\ -C_k \end{bmatrix} \tag{19.42}$$

$$F(x_k + P_k, \lambda_k + P_k) \simeq F(x_k, \lambda_k) + \nabla F(x_k, \lambda_k)[P_k, P_\lambda]^T = 0$$

$$\nabla F(x_k, \lambda_k) \begin{bmatrix} P_k \\ P_\lambda \end{bmatrix} = -F(x_k, \lambda_k)$$

equivalente a 19.42.

A iteração definida por 19.42 é conhecida como método de Newton-Lagrange, sendo bem-definida quando ∇F é uma matriz não-singular. A não-singularidade desta matriz é uma consequência das condições estabelecidas abaixo.

Condição: 19.5.1

i. A matriz Jacobiana das restrições A_k tem posto de linha completo.

ii. A matriz W_k é positiva definida no espaço tangente das restrições, i.e., $d^T w_k d > 0$ para todo $d \neq 0$ tal que $A_k d = 0$.

A premissa (1) advém da condição de cualificação de independéncia das restrições. A premissa (2) é satisfeita quando (x, λ) é próximo do ótimo (x^*, λ^*) e as condições suficientes de segunda-ordem são satisfeitas pela solução.

Sob as condições acima estabelecidas, o algoritmo de Newton 19.41 e 19.42 converge quadraticamente para o ótimo local, constituindo um ótimo algoritmo para resolver problemas não-lineares sob restrições de igualdade, isto se o ponto inicial x_0 está próximo do ótimo local x^*.

19.5.2 Linhas Gerais do Método SQP

Há uma abordagem alternativa a iteração 19.41, 19.42. Para o iterando (x_k, λ_k) definimos o problema quadrático abaixo:

$$\underset{P}{Min} \quad \tfrac{1}{2} P^T W_k P + \nabla f_k^T P \tag{19.43}$$

$$Sujeito \ \ a: \quad A_k P + C_k = 0 \tag{19.44}$$

Se as condições 19.5.1 são satisfeitas, então 19.43 possui uma solução única (P_k, μ_k) que satisfaz:

$$W_k P_k + \nabla f_k - A_k^T \mu_k = 0 \tag{19.45}$$

$$A_k P_k + C_k = 0 \tag{19.46}$$

Uma observação importante é que P_k e μ_k podem ser identificados com a solução das equações de Newton 19.42.

Se subtrairmos $A_k \ \ \lambda_k$ de ambos os lados da primera equação de 19.42, obtemos:

$$\begin{cases} W_k P_k - A_k^T P_\lambda = -\nabla f_k + A_k^T \lambda_k \\ A_k P_k = -C_k \end{cases}$$

$$\begin{cases} W_k P_k - A_k^T P_\lambda - A_k^T \lambda_k = -\nabla f_k \\ A_k P_k = -C_k \end{cases} \Rightarrow \begin{cases} W_k P_k - A_k^T (P_\lambda + \lambda_k) = -\nabla f_k \\ A_k P_k = -C_k \end{cases}$$

Mas $\lambda_{k+1} = \lambda_k + P_\lambda$, portanto obtemos:

$$\begin{bmatrix} W_k & -A_k^T \\ A_k & 0 \end{bmatrix} \begin{bmatrix} P_k \\ \lambda_{k+1} \end{bmatrix} = \begin{bmatrix} -\nabla f_k \\ -C_k \end{bmatrix} \tag{19.47}$$

Portanto, devido a não-singularidade da matriz de coeficientes, temos que $P = P_k$ e $\lambda_{k+1} = \mu_k$. Vamos nos referir a este resultado como *equivalência entre SQP e o método de Newton:* se as condições 19.5.1 são garantidas, então o próximo iterando (x_{k+1}, λ_{k+1}) pode ser definido em termos de:

i. a solução do problema quadrático 19.43 ou

ii. o iterando gerado pelo método de Newton 19.41, 19.42.

Essas interpretações alternativas são muito úteis.

A interpretação em termos do método de Newton facilita a análise, enquanto que a estrutura do SQP nos permite derivar algoritmos práticos no âmbito de restrições de igualdade.

Na sua forma mais simples, o método SQP pode ser especificado como segue:

ALGORITMO 19.5.2 (Algoritmo SQP local)

Escolha um ponto inicial (x_0, λ_0)
Para $k = 0, 1, 2, ...,$ faça
Calcule $f_k, \nabla f_k, W_k = W(x_k \lambda_k), C_k, eA_k$
Resolve o problema 19.43, obtendo $P_k e \mu_k$
$x_{k+1} \leftarrow x_k + P_k$
$\lambda_{k+1} \leftarrow \mu_k$
Se o criterio de convergência e satisfeito
Pare e retorne a solução aproximada (x_{k+1}, λ_{k+1})
Fim-se
Fim-para

É fácil de se estabelecer um resultado de convergência local para este algoritmo, uma vez que sabemos que este é equivalente ao método de Newton aplicado ao problema $F(x, \lambda) = 0$. Mais especificamente, se as condições 19.5.1 são satisfeitas pela solução (x^*, λ^*) de 19.35, se f e c são duas vezes diferenciáveis satisfazendo a condição Lipschitz, e se o ponto (x_0, λ_0) é suficientemente próximo de (x^*, λ^*),então os iterandos gerados pelo algoritmo acima convergem quadraticamente para (x^*, λ^*).

19.5.3 Restrições de desigualdade

O algoritmo SQP geral pode ser facilmente estendido para o caso geral de programação não-linear:

$$Minimize f(x) \tag{19.48}$$

$$Sujeito \ \ a:$$

$$C_i(x) = 0, \quad i \in \mathcal{E} \tag{19.49}$$

$$C_i(x) \geq 0, \quad i \in \mathcal{I} \tag{19.50}$$

Para definir o subproblema, agora linearizamos tanto as restrições de igualdade quanto as de desigualdade de forma a se obter:

$$Minimize \ \ \frac{1}{2} P^T W_k P + \nabla f_k^T P \tag{19.51}$$

$$Sujeito \ \ a:$$

$$\nabla C_i(x_k)^T P + C_i(x_k) = 0 \quad i \in \mathcal{E} \tag{19.52}$$

$$\nabla C_i(x_k)^T P + C_i(x_k) \geq 0 \quad i \in \mathcal{I} \tag{19.53}$$

Podemos então utilizar um dos algoritmos de programação quadratica existente na literatura, tal como o algoritmo de conjuntos ativos.

Um método SQP local segue portanto do algoritmo 19.5.2, com uma pequena modificação: o passo P_k e a estimativa do multiplicador de Lagrange λ_{k+1} são definidos conforme solução do problema 19.51.

O resultado a seguir mostra que esta abordagem eventualmente identifica o conjunto ativo ótimo para o problema 19.48.

Teorema 19.5.3.1 *Suponha que x^* é uma solução para 19.48. Assuma que a matriz Jacobiana A_k do conjunto de restrições ativas no ponto x^* possui posto de linha cheio, e que $d^T W_* d > 0$ para todo $d \neq 0$ tal que $A_* d = 0$, e que complementaridade estrita é garantida. Então se (x_k, λ_k) é suficientemente próximo de (x^*, λ^*), existe uma solução local do subproblema 19.51 cujo conjunto ativo A_k é idêntico ao conjunto ativo $A(x^*)$ do problema não-linear 19.48 no ponto x^*.*

19.5.4 Implementação de SQP

Existema pelo menos duas formas de se implementar SQP.

A primera abordagem resolve a cada iteração o subproblema quadrático 19.51, tomando o conjunto ativo na solução deste subproblema como uma estimativa do conjunto ativo ótimo. Essa abordagem é denotada por IQP (Inequality-Constrained QP), a qual tem tido sucesso na prática. A dificuldade está em resolver um problema quadrático a cada iteração, mas pode ser mitigado se utilizamos um bom ponto de partida para a solução ("hot-start").

A segunda abordagem seleciona um subconjunto das restrições a cada iteração para atuar como conjunto de trabalho, resolvendo apenas subproblemas com igualdades, ignorando-se as demais restrições. O conjunto de trabalho é atualizado a cada iteração através de regras baseadas nos estimadores dos multiplicadores de Lagrange. Esta abordagem, denotada por EQP, tem a vantagem de tratar apenas de subproblemas quadráticos com restrições de igualdade, o que é computacionalmente mais fácil.

19.6 Exercícios

EX 19.1: Aplique o algoritmo de barreiras para resolver o problema abaixo e ilustre graficamente as suas iterações:

$$
\begin{array}{lll}
\text{Minimize} & f(x) = -3x_1 - 2x_2 & \\
\text{Sujeito a :} & & \\
& g_1(x) = 1 - x_1 + 2x_2 & \geq \ 0 \\
& g_2(x) = 2 - x_1 + x_2 & \geq \ 0 \\
& g_3(x) = 6 - 2x_1 + x_2 & \geq \ 0 \\
& g_4(x) = 5 - x_1 & \geq \ 0 \\
& g_5(x) = 16 - 2x_1 - x_2 & \geq \ 0 \\
& g_6(x) = 12 - x_1 - x_2 & \geq \ 0 \\
& g_7(x) = 21 - x_1 - 2x_2 & \geq \ 0 \\
& g_8(x) = 10 - x_2 & \geq \ 0 \\
& g_9(x) = x_1 & \geq \ 0 \\
& g_{10}(x) = x_2 & \geq \ 0
\end{array}
$$

Passos sugeridos:

1) Encontre uma solução inicial x^0 que pertença ao interior da região factível

2) Obtenha a função barreira $b(x;\mu) = f(x) - \mu \sum_{k=1}^{10} \log(g_k(x))$

3) Execute o algoritmo abaixo:

   – Seja x^0 o iterando inicial
   – Seja μ^1 a penalidade inicial, por exemplo $\mu^1 = 10$
   – Seja $t = 1$ o número de iteração
   – Seja $\epsilon > 0$ um número próximo de zero definindo o critério de parada
   – Repita enquanto a condição de parada não for satisfeita
      * Encontre o mínimo x^t de $b(x, \mu^t)$ usando como ponto inicial x^{t-1}
      * Se $\|x^t - x^{t-1}\| < \epsilon$, então pare e retorne a solução x^t
      * Faça $\mu^{t+1} \in (0, \mu^t)$, por exemplo faça $\mu^{t+1} = 0.7\mu^t$
   – Fim-repita

Observação: se o algoritmo for implementado em Matlab, faça uso da função "`fminsearch`" e também reduza o parâmetro de tolerância "`TolX`" para 10^{-8}.

EX 19.2: Elipsóides são conjuntos convexos com amplo emprego em otimização convexa [27]. Além da conveniência computacional, eles podem ser empregados para aproximar outros conjuntos convexos ou não convexos. Uma representação padrão de um elipsóide $\mathcal{E}$ é:

$$\mathcal{E} = \{x : (x - x_0)^T A^{-1} (x - x_0)\}$$

onde $A = A^T$ é uma matriz positiva definida, e $x_0 \in \mathbb{R}^n$ é o centro do elipsóide. A Figura 19.4 dá uma ilustração de elipsóide no $\mathbb{R}^2$. Os eixos do elipsóide tem direções dadas pelos autovetores de A, com comprimento dado por $\sqrt{\lambda_i}$ onde λ_i é o autovalor associado ao i-ésimo autovetor. O volume do elipsóide é proporcional a $\sqrt{det(A)}$. Note que $\lambda_i > 0$ pois A é positiva definida. Note também que $det(A) = det(U^T \Lambda U) = det(U^T) det(\Lambda) det(U)$ onde $U^{-1} = U^T$ e $\Lambda = diag(\lambda_1, \ldots, \lambda_n)$. Logo, $det(A) = det(\Lambda) = \prod_{i=1}^{n} \lambda_i$ (pois $det(U^T) = 1/det(U)$). Em verdade, um elipsóide $\mathcal{E}$ pode ser representado equivalentemente nas formas abaixo:

1) $\mathcal{E} = \{x : (x - x_0)^T A^{-1} (x - x_0)\}$, com volume $\sqrt{det(A)}$.
2) Imagem de uma bola unitária sob uma transformação afim: $\mathcal{E} = \{Bu + x_0 : \|u\| \leq 1\}$ com volume $det(B)$.
3) Pré-imagem de uma bola unitária sob uma transformação afim: $\mathcal{E} = \{x : \|Ax - b\| \leq 1\}$ com volume $det(A^{-1})$.
4) Subnível de um conjunto quadrático degenerado: $\mathcal{E} = \{x : f(x) \leq 0\}$ onde $f(x) = x^T C x + 2d^T x + e$ com $C = C^T$ positiva definida, $e - d^T C^{-1} d < 0$, e volume $\sqrt{det(A)}$.

Mostre como se pode converter entre representações.

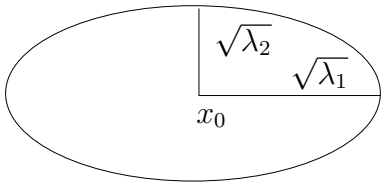


Figura 19.4: Elipsóide

Referências Bibliográficas

[1] E. Aart and J. Korst. *Simulated Annealing and Boltzmann Machines*. Wiley, New York, 1989.

[2] M. Agarwal. A systematic classification of neural-network-based control. *IEEE Control Systems Magazine*, 17(2):75–93, 1997.

[3] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows*. Prentice-Hall, Upper Saddle River, New Jersey, 1993.

[4] R. J. Aumann and S. Hart. *Handbook of Game Theory with Economic Applications*, volume 1. North-Holland, Amesterdan, 1992.

[5] D. O. Awduche, J. Malcom, J. Agogbua, J. O'Dell, and J. McManus. Requirements for traffic engineering over MPLS—request for comments, September 1999.

[6] S. N. Balakrishnan and R. D. Weil. Neurocontrol: A literature survey. *Mathematical Modeling and Computing*, 23:101–117, 1996.

[7] T. Basar and G. J. Olsder. *Dynamic Noncooperative Game Theory*. Society for Industrial and Applied Mathematics, 1999.

[8] M. S. Bazaraa, H. D. Sherali, and C. M. Shetty. *Nonlinear Programming Theory and Algorithms*. John Wiley, 1993.

[9] D. P. Bertsekas. *Nonlinear Programming*. Athena Scientific, 1995.

[10] A. E. Bryson and Y.-C. Ho. *Applied Optimal Control*. Blaisdell, 1969.

[11] V. Chvatal. *Linear Programming*. W. H. Freeman, 1983.

[12] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, Cambridge, Massachusetts, 1990.

[13] L. Costa and P. Oliveira. Evolutionary algorithms approach to the solution of mixed integer non-linear programming problems. *Computers and Chemical Enginnering*, 25:257–266, 2001.

[14] Charles Darwin. On the origin of species by means of natural selection, or the preservation of favoured races in the struggle for life. 1859.

[15] L. Davis. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, 1991.

[16] J. E. Dennis and J. J. Moré. Quasi-Newton methods, motivation and theory. *SIAM Review*, pages 46–88, 1977.

[17] R. A. Dias, E. Camponogara, J.-M. Farines, R. Willrich, and A. Campestrini. Implementing traffic engineering in MPLS-based IP networks with lagrangean relaxation. In *Proceedings of the 8th IEEE Symposium on Computers and Communications*, 2003.

[18] R. A. Dias, J.-M. Farines, E. Camponogara, R. Willrich, and A. Campestrini. Engenharia de tráfego dinâmica em redes IP com tecnologia mpls: otimização baseada em heurísticas. In *Anais do XXII Simpósio Brasileiro de Redes de Computadores*, 2004.

[19] P. J. Flemming and R. C. Purshouse. Evolutionary algorithms in control systems engineering: a survey. *Control Engineering Practice*, 10:1223–1241, 2002.

[20] R. Fourer, D. M. Gay, and B. W. Kernighan. *AMPL: A Modeling Language for Mathematical Programming*. Duxbury Press, 2002.

[21] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.

[22] Philip E. Gill, Walter Murray, and Margaret H. Wright. *Practical Optimization*. Academic Press, October 1981.

[23] M. Girish, B. Zhou, and J.Q. Hu. Formulation of the traffic engineering problems in MPLS based IP networks. In *Proceedings of the IEEE Symposium on Computers and Communications*, 2000.

[24] D. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison Wesley, 1989.

[25] C. Guéret, C. Prins, and M. Sevaux. *Applications of optimization with Xpress-MP*. Dash Optimization, Englewood Cliffs, NJ, 2002.

[26] M. Hagan and H. Demuth. Neural networks for control. In *Proceedings of the American Control Conference*, pages 1642–1656, San Diego, CA, June 1999. Invited Tutorial.

[27] H. Hindi. A tutorial on convex optimization. In *Proceedings of American Control Conference*, pages 3252–3265, Boston, Massachusetts, June 2004.

[28] J. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.

[29] K. J. Hunt, D. Sbarbaro, R. Zbikowski, and P. J. Gawthrop. Newural networks for control systems – a survey. *Automatica*, 28:1083–1112, 1992.

[30] N. Karmakar. A new polynomial-time algorithm for linear programming. *Combinatorica*, pages 373–395, 1984.

[31] S. Kirkpatrick. Optimization by simulated annealing: Quantitative studies. *Journal of Stastical Physics*, 34:975–986, 1984.

[32] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[33] S. M. LaValle. Robot motion planning: a game-theoretic foundation. *Algorithmica*, 26:430–465, 2000.

[34] E. L. Lawler. *Combinatorial Optimization: Networks and Matroids*. Holt, Rinehart and Winston, New York, 1976.

[35] M. Mesterton-Gibbons. *An Introduction to Game-Theoretic Modelling*. American Mathematical Society, 2000.

[36] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. Simulated annealing. *Journal of Chemical Physics*, 21:1087–1092, 1953.

[37] M. Minsky and S. Papert. *Perceptrons*. MIT Press, 1969.

[38] T. M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[39] J. Nocedal and S.J. Wright. *Numerical Optimization*. Springer-Verlag, 1999.

[40] D. A. Pomerlau. *Robot Learning*, chapter Knowledge-based training of artificial neural networks for autonomous robot driving, pages 19–43. Kluwer Academic Publishers, 1993.

[41] H. D. Sherali. On mixed-integer zero-one representations for separable lower-semicontinuous piecewise-linear functions. *Operations Research Letters*, 28:155–160, 2001.

[42] R. J. Vanderbei. *Linear Programming: Foundations and Extensions*. Kluwer Academic Publishers, second edition, 2001.

[43] H. P. Williams. *Model Building in Mathematical Programming*. John Wiley, 4th edition, 1999.

[44] L. A. Wolsey. *Integer Programming*. John Wiley & Sons, 1998.

[45] S. J. Wright. *Primal-Dual Interior-Point Methods*. Society for Industrial and Applied Mathematics, 1997.

[46] H. Yaiche, R. R. Mazumdar, and C. Rosenberg. A game theoretic framework for bandwidth allocation and pricing in broadband networks. *IEEE/ACM Transactions on Networking*, 8(5):667–678, October 2000.

Apêndice A

Exercícios Resolvidos

A.1 Introdução à Otimização

EX 1.1: Seja x o número de unidades do produto x a serem produzidas, enquanto y é o número de unidades do produto y. O problema pode ser colocado em programação matemática como:

$$\begin{array}{ll} \text{Maximize} & x+y \\ x,y & \\ \text{Sujeito a:} & \end{array}$$

$$\begin{array}{rcrcrcr} 50x & + & 24y & \leq & 40\times 60 & = & 2400 \\ 30x & + & 33y & \leq & 35\times 60 & = & 2100 \\ x & & & \geq & 75-30 & = & 45 \\ & & y & \geq & 95-90 & = & 5 \end{array}$$

EX 1.2: a) Basta fazer $A = \begin{bmatrix} B \\ I \end{bmatrix}$ e $b = \begin{bmatrix} d \\ 0 \end{bmatrix}$.

b) Fazendo $A = \begin{bmatrix} B \\ F \end{bmatrix}$ e $b = \begin{bmatrix} d \\ g \end{bmatrix}$ podemos reduzir o problema em consideração ao problema de mínimos quadrados.

c) Faça $A = \begin{bmatrix} B \\ \sqrt{2}F \end{bmatrix}$ e $b = \begin{bmatrix} d \\ \sqrt{2}g \end{bmatrix}$.

d) Primeiramente, a função a ser minimizada pode ser colocada na forma $\|A_1x - b_1\|^2 + \|A_2x - b_2\|^2 + \|A_3x - b_3\|^2$, onde

$$A_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \sqrt{2} & 0 \\ 0 & 0 & \sqrt{3} \end{bmatrix}, \quad b_1 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix},$$

$$A_2 = \begin{bmatrix} 1 & -1 & 1 \end{bmatrix}, \quad b_2 = [1],$$

$$A_3 = \begin{bmatrix} -1 & 4 & 0 \end{bmatrix}, \quad b_3 = [-2].$$

Por fim, basta fazer

$$A = \begin{bmatrix} A_1 \\ A_2 \\ A_3 \end{bmatrix} \text{ e } b = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

para se obter um problema de mínimos quadrados.

EX 1.3: O problema de encontrar a altura e raio da lata pode se expresso em programação não-linear com restrições:

$$\begin{array}{lll} P: & \mathit{Minimize} & f(r,h) = 2(\pi r^2 + 2\pi rh) \\ & r,h & \\ & \mathit{Sujeito\ a}: & \\ & & \pi r^2 h \geq 500 \end{array}$$

É possível eliminar a restrição de desigualdade? O problema é do tipo convexo? Pode-se obter uma solução fechada para o problema, sem fazer uso de um método de otimização?

EX 1.4: Seja $(x_m, y_m), m = 1, \ldots, M$, as coordenadas das caixas de abastecimento e seja y_t a linha onde o tubo mestre será implantado. Em programação matemática, o problema pode ser colocado como:

$$\underset{y_t}{\text{Minimize}} \quad f(y_t) = \sum_{m=1}^{M} |y_t - y_m|$$

Quais são as dificuldades para se resolver o problema acima? É possível transformá-lo em um problema de programação linear? Será que a solução y_t^* obtida por mínimos quadrados, em vez de norma-1, resultará na mesma solução— a solução por mínimos quadrados consiste na minimização da função $f(y_t) = \sum_{m=1}^{M}(y_m - y_t)^2$.

EX 1.5: Seja $x_{ij} = 1$ se a posição (i, j) do tabuleiro é ocupada por uma rainha e, caso contrário, $x_{ij} = 0$. Seja $n = 8$ o número de colunas e também de linhas do tabuleiro. O problema pode então ser colocado em programação linear inteira:

$$\begin{aligned}
&\text{Maximize} && \sum_{i=1}^{n}\sum_{j=1}^{n} x_{ij} \\
&\text{Subjeito a:} \\
&&\sum_{i=1}^{n} x_{ij} \leq 1, && j = 1, \ldots, n \\
&&\sum_{j=1}^{n} x_{ij} \leq 1, && i = 1, \ldots, n \\
&&\sum_{i=1}^{n-k} x_{i,i+k} \leq 1 && k = 0, \ldots, n-2 \\
&&\sum_{i=1}^{n-k} x_{i+k,i} \leq 1 && k = 1, \ldots, n-2 \\
&&\sum_{i=1}^{n-k} x_{i,n-i+1-k} \leq 1 && k = 0, \ldots, n-2 \\
&&\sum_{i=1}^{n-k} x_{i+k,n-i+1} \leq 1 && k = 1, \ldots, n-2 \\
&&x_{ij} \in \{0, 1\} && i = 1, \ldots, n \\
&&&& j = 1, \ldots, n
\end{aligned}$$

EX 1.6: Sendo $z(x, y)$ o vetor de decisão, o problema é encontrar z tal que:

$$\begin{aligned}
\|z - z_1\| &= d_1 \\
\|z - z_2\| &= d_2
\end{aligned}$$

Equivalentemente, podemos expressar o problema como:

$$\begin{aligned}
\|(x, y) - (x_1, y_1)\|^2 &= d_1^2 \\
\|(x, y) - (x_2, y_2)\|^2 &= d_2^2
\end{aligned}$$

A.2 Minimização de Funções com o Método de Descenso

EX 2.1: Pode-se verificar que:

$$\begin{aligned}\nabla f(x) &= \begin{bmatrix} -400(x_2 - x_1^2)x_1 - 2 + 2x_1 \\ 200(x_2 - x_1^2) \end{bmatrix} \\ &= \begin{bmatrix} 400x_1^3 - 400x_1x_2 - 2 + 2x_1 \\ 200x_2 - 200x_1^2 \end{bmatrix}\end{aligned}$$

e que

$$\nabla^2 f(x) = \begin{bmatrix} 1200x_1^2 - 400x_2 + 2 & -400x_1 \\ -400x_1 & 200 \end{bmatrix}$$

Observando que $f(x) \geq 0$ para qualquer x, o menor valor que esta pode assumir é $f(x^*) = 0$. Uma vez que $x^* = (1, 1)$ é tal que $f(x^*) = 0$, concluímos que x^* é um ótimo global. Note que para $x_1 \neq 1$ ou $x_2 \neq x_1^2$ tem-se $f(x) > 0$, portanto x^* é único. Pode-se calcular o gradiente e a Hessiana no ponto x^*, obtendo:

$$\nabla f(x^*) = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \text{ e } \nabla^2 f(x^*) = \begin{bmatrix} 802 & -400 \\ -400 & 200 \end{bmatrix}$$

Logo, x^* satisfaz as condições necessárias e suficientes pois $\nabla f(x^*) = 0$ e $\nabla^2 f(x^*)$ é positiva definida.

EX 2.2: Primeiramente, obtemos o gradiente de f:

$$\nabla f(x) = \begin{bmatrix} 8 + 2x_1 \\ 12 - 4x_2 \end{bmatrix} = 0 \Leftrightarrow x_1 = -4 \text{ e } x_2 = 3.$$

Dessa forma, $x^* = (-4, 2)$ é o único ponto que satisfaz as condições de otimalidade de primeira ordem. Calculando a Hessiana neste ponto, obtemos

$$\nabla^2 f(x^*) = \begin{bmatrix} 2 & 0 \\ 0 & -4 \end{bmatrix}$$

Sendo $\nabla^2 f(x^*)$ indefinida, concluímos que x^* não é um ponto máximo nem mínimo local.

EX 2.3: $\nabla f_1(x) = c$ e $\nabla^2 f_1(x) = \mathbf{0}$. $\nabla f_2(x) = \frac{1}{2}Ax + \frac{1}{2}A^T x$ e $\nabla^2 f_2(x) = \frac{1}{2}A^T + \frac{1}{2}A$. Se A for simétrica, então $\nabla f_2(x) = Ax$ e $\nabla^2 f_2(x) = A$.

Como exemplo, seja $f_2(x)$ dada pela matriz:

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$$

Então fica fácil de verificar que:

$$\begin{aligned}
\nabla f_2(x) &= \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} a_{11} & a_{21} \\ a_{12} & a_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \\
&= \begin{bmatrix} 2a_{11} & a_{12}+a_{21} \\ a_{12}+a_{21} & 2a_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \text{ e} \\
\nabla^2 f_2(x) &= \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} + \begin{bmatrix} a_{11} & a_{21} \\ a_{12} & a_{22} \end{bmatrix} \\
&= \begin{bmatrix} 2a_{11} & a_{12}+a_{21} \\ a_{12}+a_{21} & 2a_{22} \end{bmatrix}
\end{aligned}$$

EX 2.4: a) $h(\alpha x_1+(1-\alpha)x_2) = f(\alpha x_1+(1-\alpha)x_2)+g(\alpha x_1+(1-\alpha)x_2) \leq \alpha f(x_1)+(1-\alpha)f(x_2)+\alpha g(x_1)+(1-\alpha)g(x_2) = \alpha h(x_1)+(1-\alpha)h(x_2)$, logo $h(x)$ é convexa.

b) $h(x) = f(x) - g(x)$ pode não ser convexa. Para $f(x) = x$ e $g(x) = x^2$ obtemos $h(x)$ côncava.

c) $h(x) = f(x)^2$ não é necessariamente convexa. Para $f(x) = x^2 - 30$ pode-se verificar que $h(x)$ não é convexa.

d) Para $f(x) = x$ verifica-se $f(x) \geq 0$ no domínio $x \geq 0$. Contudo, $h(x)$ é uma função concava no domínio $x \geq 0$.

EX 2.5: Para o problema definido por

$$Q = \begin{bmatrix} 0,9561 & 0,6888 \\ 0,6888 & 0,6044 \end{bmatrix} \text{ e } b = \begin{bmatrix} 1 \\ -2 \end{bmatrix},$$

o algoritmo de descenso produziu a seguinte lista iterandos

$$\begin{array}{rr}
54,5022 & 30,3585 \\
6,1911 & -8,4746 \\
19,3641 & -24,8628 \\
19,1041 & -25,0717 \\
19,1750 & -25,1599 \\
19,1736 & -25,1611 \\
19,1739 & -25,1615 \\
19,1739 & -25,1615
\end{array}$$

que está ilustrada na Figura **??**. Na implementação, o passo do algoritmo foi calculado de forma ótima, simplesmente fazendo a derivada da função $f(x_k + \alpha_k p_k) = f(\alpha_k)$ igual zero.

EX 2.7: Seja $A = (I-\alpha Q)^2 \geq 0$. Note que $A = A^T$ e, portanto, existe uma fatorização $A = U^T \Lambda U$ onde $U^{-1} = U^T$, $\Lambda = diag(\lambda_1, \ldots, \lambda_n)$ e $\lambda_i, i = 1, \ldots, n$, são os

autovalores de A. Logo, temos:

$$\begin{aligned}
x^T(I-\alpha Q)^2x &= x^TAx \\
&= x^TU^T\Lambda Ux \\
&= z^T\Lambda z \qquad\qquad [z=Ux] \\
&= \sum_{i=1}^{n} z_i^2\lambda_i \\
&\leq z_i^2 \max\{\lambda\in\sigma(A)\} \\
&= \max\{\lambda\in\sigma(A)\}z^Tz \\
&= \max\{\lambda\in\sigma(A)\}(x^TU^TUx) \\
&= \max\{\lambda\in\sigma(A)\}\|x\|^2 \\
&= \max\{\lambda\in(I-\alpha Q)^2\}\|x\|^2
\end{aligned}$$

EX 2.8: Fazendo $Q = U^T\Lambda U$ onde $\Lambda = diag(\sigma(Q))$, basta verificar que $(I-\alpha_kQ)^2 = (I-\alpha_kU^T\Lambda U)^2 = (U^TU-\alpha_kU^T\Lambda U)^2 = [U^T(I-\alpha_k\Lambda)U]^2 = U^T(I-\alpha_k\Lambda)UU^T(I-\alpha_k\Lambda)U = U^T(I-\alpha_k\Lambda)^2U$. Portanto, os autovalores de $(I-\alpha_kQ)^2)$ são iguais a $(1-\alpha_k\lambda_i)^2$.

EX 2.12: As respostas são dadas abaixo.

i. Prof. Kunz está correto. Note que $f'(x) = 2/(e^x-e^{-x})^2$. Observe ainda que $f'(x) = 0$ apenas quando $x\to-\infty$ ou $x\to\infty$, ou seja, $\lim_{x\to-\infty}f'(x) = \lim_{x\to\infty}f'(x) = 0$. Logo não existe mínimo local pois $f'(x) = 0$ é condição necessária.

ii. Correto, pois todo mínimo local x de uma função contínua diferenciável deve satisfazer a condição necessária $\nabla f(x) = 0$.

iii. Prof. Kunz se equivocou. Considere a função $f(x) = x^3 - 4x^2 - 10x$. Apesar de f admitir um mínimo local $x = 3.594118$, $\lim_{x\to-\infty}f(x) = -\infty$. Concluímos que f não admite um ponto de ótimo global; a função é ilimitada por baixo.

iv. A afirmação está incorreta. Para $f(x) = x^3$ tem-se $f'(x) = 3x^2$ e $f''(x) = 6x$. Para $x = 0$ verificamos que $f'(x) = 0$ e $f''(x) = 0$ (x satisfaz as condições necessárias de segunda ordem) contudo x não é um mínimo local. Para qualquer $\Delta x > 0$ podemos verificar que $f(x-\Delta x) < f(x)$ e, daí, deduzimos que existe $x' = x-\Delta x$ na vizinhança de x com valor inferior.

v. Prof. Kunz está correto. A condição $f''(x)\geq 0$ para todo x implica f ser convexa. Uma vez que as condições necessárias de primeira ordem são também suficientes concluímos que a afirmação procede.

vi. Observe que $f'(x) = -\left(\frac{x+2}{x^3}\right)e^{-x}$ e $f''(x) = \frac{6+4x+x^2}{x^4}e^{-x}$. Note que para $x\in R = [1,\infty)$, $f''(x) > 0 \Rightarrow f$ é convexa. Assim, se existir um mínimo local $x\in R$ então obrigatoriamente x é um mínimo global. Prof. Kunz está correto.

vii. Prof. Kunz se equivocou nesta afirmação. Seja $f(x,y) = x^2 + y^2$ e $g(x,y) = x + y$ o que implica $h(x,y) = x^2 + y^2 + x + y$. Note que h é convexa e podemos encontrar um mínimo global $(\hat{x}, \hat{y})$ fazendo:

$$\nabla h(\hat{x}, \hat{y}) = \begin{bmatrix} 2\hat{x} + 1 \\ 2\hat{y} + 1 \end{bmatrix} = 0 \Leftrightarrow \begin{cases} \hat{x} = -1/2 \\ \hat{y} = -1/2 \end{cases}$$

Apesar de $(\hat{x}, \hat{y})$ ser um mínimo global para h, $(\hat{x}, \hat{y})$ não é mínimo local de f, muito menos ainda um mínimo de g que não tem mínimo local.

A.3 Método de Newton

EX 3.1: Para encontrarmos as regiões de convergência e divergência podemos, inicialmente, descobrir a região em que o método de Newton tem comportamento cíclico. Para função $f(x) = \arctan(x)$, o operador de Newton é definido por

$$x_{k+} = x_k - \frac{f(x_k)}{f'(x_k)} = x_k - \frac{\arctan(x_k)}{1/(1+x_k^2)} \tag{A.1}$$

Seja x um iterando qualquer e x^+ o iterando seguinte. Para que o método de Newton entre em laço infinito, se faz necessário que $|x^+ - x^*| = |x - x^*|$ sendo $f(x^*) = 0$. Há duas possibilidades: (i) $x^+ - x^* = x - x^*$, que implica $x^+ = x$ e portanto x define uma raiz; (ii) $x^+ - x^* = -(x - x^*)$. Vamos então analisar o segundo caso, quando $x^+ - x^* = -x + x^*$. Substituindo (A.1) e sabendo que $x^* = 0$, deduzimos que:

$$x - (1+x^2)\arctan(x) = -x \Leftrightarrow 2x - (1+x^2)\arctan(x) = 0 \tag{A.2}$$

Portanto, um ponto em que o método de Newton entra em ciclo infinito é precisamente uma solução para (A.2). As soluções de (A.2) são -1.39174520027073, 0 e -1.39174520027073. Logo, o método de Newton diverge para $x_0 \in (-\infty, -1.39174520027073) \cup (1.39174520027073, \infty)$ e converge para $x_0 \in (-1.39174520027073, 1.39174520027073)$. O método entra em ciclo infinito para $x_0 \in \{1.39174520027073, 1.39174520027073\}$.

EX 3.2: Com $x^{(0)} = (1,1)$ o método converge para $x = (0.69684555124075, 0.28559372228403)$. Com $x^{(0)} = (-1,1)$ o método converge para $x = (-0.69684555124075, 0.28559372228403)$. Com $x^{(0)} = (1,-1)$ e $x^(0) = (-1,-1)$ o método não converge. As soluções do sistema de equações não-lineares estão ilustradas na Figura A.1.

EX 3.3: Seja $F(x) = 0$ o sistema de equações não-lineares no espaço das variáveis x. A partir de um iterando inicial x^0, o método de Newton é dado por:

$$x^{k+1} = x^k - \nabla F(x^k)^{-1} F(x^k), \quad k = 0, 1, \ldots$$

Seja $x = H(y)$ uma função de y onde $H(y) = Sy$. A solução do sistema F pode ser colocado no espaço de y como:

$$G(y) = 0, \text{ onde } G(y) = F(H(y))$$

Portanto,

$$\nabla G(y) = \nabla F(H(y) = \nabla_x F(x).\nabla_y H(y) = \nabla F(x).S$$

O método de Newton em y fica:

$$\begin{aligned}
y^{k+1} &= y^k - \nabla G(y^k)^{-1} G(y) \\
&= y^k - [\nabla F(Sy^k)S]^{-1} F(Sy^k) \\
&= y^k - S^{-1}\nabla F(Sy^k)^{-1} F(Sy^k) \\
&= S^{-1}x^k - S^{-1}\nabla F(x^k)^{-1} F(x^k) \qquad \text{[Por indução } x^k = Sy^k\text{]} \\
&= S^{-1}[x^k - \nabla F(x^k)^{-1} F(x^k)] \\
&= S^{-1}x^{k+1}
\end{aligned}$$

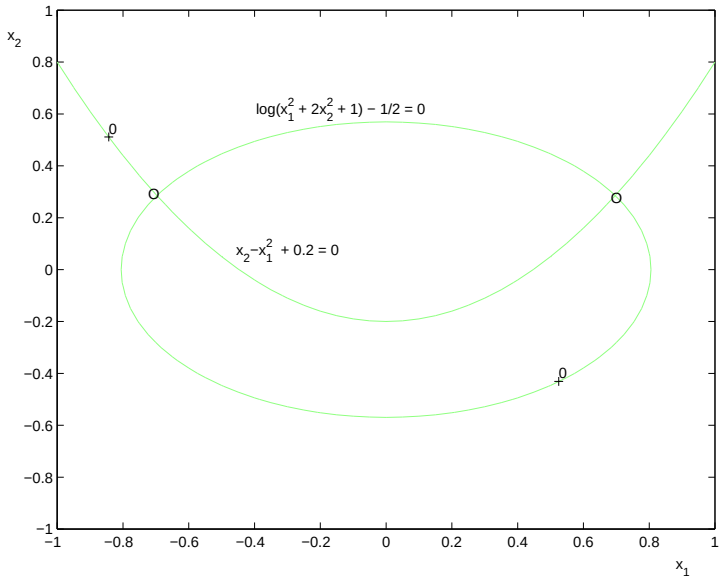


Figura A.1: Soluções do sistema de equações não-lineares

Logo $x^{k+1} = Sy^{k+1}$ como se esperava.

EX 3.4: Seja $\bar{x}$ uma solução do sistema de equações lineares. Existe solução pois $rank(A) = m$. Seja $B \in \mathbb{R}^{m \times n-m}$ uma base do espaço nulo de A. Então podemos escrever qualquer solução x como $x = \bar{x} + By$, o que nos leva a colocar P na forma abaixo:

$$P': \begin{array}{ll} \text{Minimize} & f(y) = \|\bar{x} + By - c\|^2 \\ y \in \mathbb{R}^{n-m} & \end{array}$$

Logo, reduzimos P a um problema P' sem restrições.

Note que $\nabla f(y) = 2B^T\bar{x} + 2B^TBy - 2B^Tc$ e $\nabla^2 f(y) = 2B^TB$. Uma vez que B tem posto de coluna completo, concluímos que $\nabla^2 f(y)$ é positiva definida. Logo, todo ponto estacionário satisfaz as condições suficientes de otimalidade de segunda ordem e, uma vez que $f(y)$ é convexa, concluímos que todo ponto estacionário define um ótimo global. Para encontrar um ótimo global basta resolver o sistema de equações lineares $\nabla f(y) = 0$.

EX 3.5: As raízes da equação são $r_1 = 1,259921$, $r_2 = -0,629961 - 1,09112i$ e $r_3 = -0,629961 + 1,09112i$. O fractal obtido com o procedimento está ilustrado na Figura **??**.

EX 3.6: (x, y) define um ponto estacionário se $\nabla f(x, y) = 0$. Logo,

$$\nabla f(x,y) = \begin{bmatrix} 2x + \beta y + 1 \\ 2y + \beta x + 2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{A.3}$$

o que nos leva a concluir que:

$$\begin{aligned} x &= \frac{2(\beta-1)}{4-\beta^2} \\ y &= \frac{\beta-4}{4-\beta^2} \end{aligned}$$

Portanto, dado $|\beta| \neq 2$ o ponto estacionário é uma função de β dado por:

$$\begin{bmatrix} x(\beta) \\ y(\beta) \end{bmatrix} = \begin{bmatrix} \frac{2(\beta-1)}{4-\beta^2} \\ \frac{\beta-4}{4-\beta^2} \end{bmatrix}$$

Para $\beta \in \{2, -2\}$ o sistema de equações (A.3) não tem solução e não há ponto estacionário.

Para verificar quais desses pontos são mínimos locais, precisamos verificar a Hessiana de f nos pontos estacionários. A Hessiana é dada por:

$$\nabla^2 f = \begin{bmatrix} 2 & \beta \\ \beta & 2 \end{bmatrix}$$

Para $(x(\beta), y(\beta))$ ser um ponto estacionário, este deve satisfazer as condições necessárias de segunda ordem. Isto significa que $\nabla^2 f(x(\beta), y(\beta))$ deve ser positiva semi-definida, em outras palavras, os autovalores de $\nabla^2 f$ devem ser não negativos. Para tanto, $\nabla^2 f > 0 \Leftrightarrow |\beta| \leq 2$.

Uma vez que para $|\beta| = 2$ não existe ponto estacionário, todo mínimo local $(x(\beta), y(\beta))$ satisfaz $|\beta| < 2$ e, consequentemente, satisfaz as condições suficientes de segunda ordem. Além disso, $f(x)$ se torna convexa e todo ponto estacionário é um mínimo local que por sua vez é um mínimo global.

EX 3.8: Analisamos cada uma das funções separadamente.

1) Observando que $\nabla f(x, y) = (4x(x^2 - 4), 2y)$ podemos calcular os pontos estacionários fazendo $\nabla f = 0$. Daí concluímos que os pontos estacionário são $z_1 = (0, 0), z_2 = (2, 0), z_3 = (-2, 0)$. Observando que $f(x, y) \geq 0$ para todo (x, y), podemos concluir que z_2 e z_3 são mínimos globais.
   Calculando a Hessiana de f, obtemos

$$\nabla^2 f = \begin{bmatrix} 12x^2 - 16 & 0 \\ 0 & 2 \end{bmatrix}$$

   No ponto $(0, 0)$ verificamos que $\nabla^2 f(0, 0)$ é indefinida implicando que $(0, 0)$ não é um mínimo local, nem um mínimo global.
   A função f é ilimitada por cima e não possui máximo local.

2) Calculando o gradiente de f obtemos

$$\nabla f = \begin{bmatrix} -4x(y - x^2)^2 - 2x \\ 2(y - x^2) \end{bmatrix}$$

Forçando $\nabla f(x, y) = 0$ descobrimos que $y = x^2 \Rightarrow x = 0$ e $y = 0$. O único ponto estacionário é $(0, 0)$. A partir da Hessiana

$$\nabla^2 f = \begin{bmatrix} 12x^2 - 4y - 2 & -4x \\ -4x & 2 \end{bmatrix}$$

é possível verificar que no ponto $(0, 0)$ a Hessiana:

$$\nabla^2 f(0, 0) = \begin{bmatrix} -2 & 0 \\ 0 & 2 \end{bmatrix}$$

é indefinida. Com isso $(0, 0)$ não constitui ponto de mínimo nem máximo local.

A.4 Otimização Black-Box

EX 4.2: O problema da bi-partição máxima é essencialmente o problema de corte máximo em grafos que foi demonstrado, na sua versão de decisão, ser NP-Completo. Portanto, o problema da bi-partição máxima é NP-Difícil.

EX 4.3: O problema de corte mínimo entre dois vérticies s e t pode ser resolvido com um algoritmo de fluxo máximo. O fluxo máximo que pode ser enviado de s para t é precisamente a capacidade do corte mínimo $(S, V - S)$ entre s e t ($s \in S$ e $t \in V - S$). Uma vez que o corte mínimo em G tem $s \in S$ e outro vértice $t \in V - S$, $t \neq s$, podemos calcular o corte mínimo resolvendo $|V| - 1$ problemas de fluxo máximo: para cada $t \in V - \{s\}$, calcule o corte mínimo $C_t = (S_t, V - S_t)$ entre s e t; seja $t^* = \text{Argmin}\{w(C_t) : t \in V - \{s\}\}$. O corte mínimo em G é $C^* = C_{t^*}$.

Uma vez que o problema de fluxo máximo pode ser resolvido em tempo polinomial, concluímos que o problema da bi-partição mínima pode ser resolvido em tempo polinomial. Logo este problema pertence a classe P.

EX 4.4: Qualquer algoritmo para os problemas da bi-partição máxima e mínima pode ser empregado para resolver o respectivo problema quando os pesos são variáveis aleatórias. Basta observar que

$$E[w(S, V - S)] = E[\sum_{(u,v)\in(S,V-S)} w(u, v)] = \sum_{(u,v)\in(S,V-S)} E[w(u, v)].$$

Ou seja, simplesmente substituímos $w(u, v)$ por $E[(u, v)]$.

A.5 Treinamento de Redes Neurais

EX 5.1: Desejamos implementar as operações lógica *AND*, *NAND*, *OR* e *NOR* para entradas lógicas x_1 e x_2. Para tanto, basta definir os pesos w_0, w_1 e w_2 como segue:

AND: $w_0 = -1.5$, $w_1 = 1$ e $w_2 = 1$. A saída do perceptron será 1 somente se $w^T x = -1.5 + x_1 + x_2 > 0 \Rightarrow x_1 = x_2 = 1$; caso contrário a saída será -1.

NAND: $w_0 = 1.5$, $w_1 = -1$ e $w_2 = -1$. A saída do perceptron será 1 somente se $w^T x = 1.5 - x_1 - x_2 > 0 \Rightarrow x_1 x_2 = 0$; caso contrário a saída será 1.

OR: $w_0 = 0$, $w_1 = 1$ e $w_2 = 1$.

NOR: $w_0 = 0.5$, $w_1 = -1$ e $w_2 = -1$.

EX 5.2: Seja T o número de exemplos de treinamento. O problema de treinar a rede neural pode ser colocado em programação matemática como segue:

$$
\begin{array}{lll}
P: \text{ Minimize } & \displaystyle\sum_{t=1}^{T}\sum_{k=1}^{4}(y_k^t - o_{k+6}^t)^2 & \\
\text{Sujeito a:} & & \\
& o_k^t = x_k^t & k = 1,\ldots,4 \\
& & t = 1,\ldots,T \\
& net_k^t = \sum_{m=1}^{4} w_{m,k} o_m^t & k = 5,6 \\
& & t = 1,\ldots,T \\
& net_k^t = w_{5,k} o_5^t + w_{6,k} o_6^t & k = 7,\ldots,10 \\
& & t = 1,\ldots,T \\
& o_k^t = \sigma(net_k^t) & k = 5,\ldots,10 \\
& & t = 1,\ldots,T
\end{array}
$$

Para qualquer exemplo de treinamento $t \in \{1,\ldots,T\}$, temos que:

$$
\begin{array}{rcll}
net_k^t & = & \sum_{m=1}^{4} w_{m,k} x_m^t & k = 5,6 \\
o_k^t & = & \sigma(net_k^t) & k = 5,6 \\
 & = & \sigma(\sum_{m=1}^{4} w_{m,k} x_m^t) & \\
o_k^t & = & \sigma(net_k^t) & k = 7,\ldots,10 \\
 & = & \sigma(w_{5,k} o_5^t + w_{6,k} o_6^t) & \\
 & = & \sigma(w_{5,k}\sigma(\sum_{m=1}^{4} w_{m,5} x_m^t) + w_{6,k}\sigma(\sum_{m=1}^{4} w_{m,6} x_m^t)) &
\end{array}
$$

Logo, P pode ser colocado como um problema de otimização irrestrita:

$$
\text{Min } \sum_{t=1}^{T}\sum_{k=1}^{4}\left[y_k^t - \sigma\left(w_{5,k+6}\sigma(\sum_{m=1}^{4} w_{m,5} x_m^t) + w_{6,k+6}\sigma(\sum_{m=1}^{4} w_{m,6} x_m^t)\right)\right]^2
$$

cujas variáveis são representadas pelo vetor: $w = [w_{k,5}, w_{k,6} : k = 1,\ldots,4] \cup [w_{5,k}, w_{6,k} : k = 7,\ldots,10] \in \mathbb{R}^{16}$.

A.6 Programação Linear

EX 6.1: Uma vez que $b_1 < 0$, vamos tomar o problema auxiliar para encontrar uma solução factível:

$$\begin{array}{ll} \text{Maximize} & -x_0 \\ \text{Sujeito a :} & \\ & \begin{array}{rcrcrcrcr} -x_1 & - & x_2 & - & x_3 & - & x_0 & \leq & -2 \\ 2x_1 & - & x_2 & + & x_3 & - & x_0 & \leq & 1 \end{array} \\ & x_0, x_1, x_2, x_3 \geq 0 \end{array}$$

Tomando o dicionário inicial como se os elementos b_i's fossem todos não negativos, obtemos:

$$\begin{array}{rrcrrrrr} \text{Max} & \delta & = & & & & & -x_0 \\ & w_1 & = & -2 & +x_1 & +x_2 & +x_3 & +x_0 \\ & w_2 & = & 1 & -2x_1 & +x_2 & -x_3 & +x_0 \end{array}$$

Realizando pivoteamento em x_0 e na restrição mais negativa, temos que $x_0 = w_1 + 2 - x_1 - x_2 - x_3$ e substituindo no dicionário obtemos:

$$\begin{array}{rrcrrrrr} \text{Max} & \delta & = & -2 & +x_1 & +x_2 & +x_3 & -w_1 \\ & x_0 & = & 2 & -x_1 & -x_2 & -x_3 & +w_1 \\ & w_2 & = & 3 & -3x_1 & & -2x_3 & +w_1 \end{array}$$

que passa ser factível para o problema auxiliar. Fazendo x_1 entrar na base, a variável w_2 deve sair da base pois $3/3 = 1 < 2/1 = 1$. Logo, $x_1 = 1 - 2x_3/3 + w_1/3 - w_2/3$ e substituindo no dicionário acima obtemos o dicionário:

$$\begin{array}{rrcrrrrr} \text{Max} & \delta & = & -1 & x_2 & +x_3/3 & -2w_1/3 & -w_2/3 \\ & x_0 & = & 1 & -x_2 & -x_3/3 & +2w_1/3 & +w_2/3 \\ & x_1 & = & 1 & & -2x_3/3 & +w_1/3 & -w_2/3 \end{array}$$

Fazendo x_2 entrar na base, a variável x_0 deve sair da base. Sendo $x_2 = 1 - x_0 - x_3/3 + 2w_1/3 + w_2/3$, fazemos as devidas substiuições para obter o dicionário:

$$\begin{array}{rrcrrrrr} \text{Max} & \delta & = & & & & & -x_0 \\ & x_1 & = & 1 & -2x_3/3 & +w_1/3 & -w_2/3 & \\ & x_2 & = & 1 & -x_3/3 & +2w_1/3 & +w_2/3 & -x_0 \end{array}$$

Note que o dicionário corrente é factível para o problema original, logo basta remover a coluna correspondente à variável auxiliar x_0, substituir a função objetivo original e continuar deste ponto em diante. Calculamos a função objetivo como segue:

$$\begin{array}{rcl} f & = & 2x_1 - 6x_2 \\ & = & 2(1 - 2x_3/3 + w_1/3 - w_2/3) - 6(1 - x_3/3 + 2w_1/3 + w_2/3) \\ & = & 2 - 4x_3/3 + 2w_1/3 - 2w_2/3 - 6 + 6x_3/3 - 12w_1/3 - 6w_2/3 \\ & = & -4 + 2x_3/3 - 10w_1/3 - 8w_2/3 \end{array}$$

O dicionário para o problema original toma a forma:

$$\begin{array}{rrcrrrr} \text{Max} & \delta & = & -4 & +2x_3/3 & -10w_1/3 & -8w_2/3 \\ & x_1 & = & 1 & -2x_3/3 & +w_1/3 & -w_2/3 \\ & x_2 & = & 1 & -x_3/3 & +2w_1/3 & +w_2/3 \end{array}$$

A única variável que deve entrar na base é x_3. A variável que sai da base é x_1 pois $3/2 < 3$. Substituindo $x_3 = 3/2 - 3x_1/2 + w_1/2 - w_2/2$ na linha da função objetivo e na segunda última do dicionário, teremos

$$\begin{aligned} \delta &= 4 + 2x_3/3 - 10w_1/3 - 8w_2/3 \\ &= -4 + 2(3/2 - 3x_1/2 + w_1/2 - w_2/2)/3 - 10w_1/3 - 8w_2/3 \\ &= -4 + 1 - x_1 + w_1/3 - w_2/3 - 10w_1/3 - 8w_2/3 \\ &= -3 - x_1 - 3w_1 - 3w_2 \\ x_2 &= 1 - x_3/3 + 2w_1/3 + w_2/3 \\ &= 1 - (3/2 - 3x_1/2 + w_1/2 - w_2/2)/3 + 2w_1/3 + w_2/3 \\ &= 1 - 1/2 + x_1/2 - w_1/6 + w_2/6 + 2w_1/3 + w_2/3 \\ &= 1/2 + x_1/2 + w_1/2 + w_2/2 \end{aligned}$$

Portanto o dicionário fica:

$$\begin{array}{rrcrrrr} \text{Max} & \delta & = & -3 & -x_1 & -3w_1 & -3w_2 \\ & x_3 & = & 3/2 & -3x_1/2 & +w_1/2 & -w_2/2 \\ & x_2 & = & 1/2 & +x_1/2 & +w_1/2 & +w_2/2 \end{array}$$

O dicionário acima induz uma solução ótima, dada por:

$$x_1 = 0,\ x_2 = 1/2,\ x_3 = 3/2$$

cujo valor para a função objetivo é -3.

EX 6.2: A solução ótima é $x_1 = 4$ e $x_2 = 8$ com função objetivo $f(x) = 28$.

EX 6.4: Considere o problema de programção linear dado por:

$$\begin{array}{ll} \text{Maximize} & x_1 + x_2 \\ \text{Sujeito a :} & \\ & 3x_1 + x_2 \leq 6 \\ & x_1, x_2 \geq 0 \end{array}$$

O dicionário inicial é:

$$\begin{array}{rrcrrr} D_1: & \delta & = & 0 & +x_1 & +x_2 \\ & w_1 & = & 6 & -3x_1 & -x_2 \end{array}$$

Fazendo x_1 entrar na base, obtemos o dicionário:

$$\begin{array}{rrcrrr} D_2: & \delta & = & 2 & -w_1/3 & +2x_2/3 \\ & x_1 & = & 2 & -w_1/2 & -x_2/3 \end{array}$$

Por fim, fazendo x_2 entrar na base obrigatoriamente x_1 sai da base, resultando no dicionário:

$$\begin{array}{rrcrrr} D_3: & \delta & = & 6 & -w_1 & -2x_1 \\ & x_2 & = & 6 & -w_1 & -3x_1 \end{array}$$

EX 6.5: Seja x_i a variável que saiu da base e x_k a variável que entrou na base. Denote por B o conjunto dos índices das variáveis básicas; e N o conjunto dos índices das variáveis não básicas. Então, a seguinte equação pertencia ao dicionário:

$$x_i = \bar{b}_i - \sum_{j \in N} \bar{a}_{ij} x_j = \bar{b}_i - \sum_{j \in N - \{k\}} \bar{a}_{ij} x_j - \bar{a}_{ik} x_k$$

Portanto:

$$x_k = \bar{b}_i / \bar{a}_{ik} - \sum_{j \in N - \{k\}} \bar{a}_{ij} x_j - x_i / \bar{a}_{ik}$$

onde $\bar{b}_i \geq 0$ e $\bar{a}_{ik} > 0$. Sabemos também que o custo reduzido de x_k é $c_k > 0$. (Caso contrário, x_k não entraria na base.) No dicionário resultante, a variável x_i terá custo reduzido dado por:

$$c_i = -c_k / \bar{a}_{ik} < 0$$

implicando que x_i não é uma variável candidata para entrar na base na próxima iteração.

EX 6.6: O problem pode ser colocado na forma:

$$\begin{array}{lll} P: & \text{Maximize} & c^T x \\ & \text{Sujeito a :} & \\ & & \begin{array}{rcl} Ax & \leq & b \\ -Ax & \leq & -a \\ Ix & \leq & u \\ -Ix & \leq & -l \\ x & \geq & 0 \end{array} \end{array}$$

Assumindo que $A \in \mathbb{R}^{m \times n}$ e utilizando multiplicadores duais $w_1, w_2 \in \mathbb{R}^m$ e $v_1, v_2 \in \mathbb{R}^n$, obtemos o problema dual:

$$\begin{array}{lll} D: & \text{Minimize} & w_1^T b - w_2^T a + v_1^T u - v_2^T l \\ & \text{Sujeito a :} & \\ & & w_1^T A - w_2^T A + v_1^T I - v_2^T I \geq c^T \\ & & w_1, w_2, v_1, v_2 \geq 0 \end{array}$$

que pode ser expresso como:

$$\begin{array}{lll} D: & \text{Minimize} & b^T w_1 - a^T w_2 + u^T v_1 - l^T v_2 \\ & \text{Sujeito a :} & \\ & & A^T w_1 - A^T w_2 + v_1 - v_2 \geq c \\ & & w_1, w_2, v_1, v_2 \geq 0 \end{array}$$

EX 6.7: Resolvendo o dual:

$$\begin{array}{lll} D: & \text{Minimize} & -2y_1 + y_2 \\ & \text{Sujeito a :} & \\ & & \begin{array}{rrcl} -y_1 & +2y_2 & \geq & 2 \\ -y_1 & -y_2 & \geq & -6 \\ -y_1 & +y_2 & \geq & 0 \\ y_1 & & \geq & 0 \\ & y_2 & \geq & 0 \end{array} \end{array}$$

obtemos a solução $y_1 = 3$ e $y_2 = 3$. Aplicando o Teorema da Folga Complementar, podemos verificar que:

$$\begin{aligned}
(-y_1 + 2y_2 - 2)x_1 &= (-3 + 2 \times 3 - 2)0 \\
&= 1 \times 0 = 0 \\
(-y_1 - y_2 + 6)x_2 &= (-3 - 3 + 6)1/2 \\
&= 0 \times 1/2 = 0 \\
(-y_1 + y_2 - 0)x_3 &= (-3 + 3 - 0)3/2 = 0 \\
(-x_1 - x_2 - x_3 + 2)y_1 &= (0 - 1/2 - 3/2 + 2)3 \\
&= 0 \times 3 = 0 \\
(-2x_1 - x_2 + x_3 - 1)y_2 &= (-2 \times 0 - 1/2 + 3/2 - 1)3 \\
&= 0 \times 3 = 0
\end{aligned}$$

EX 6.9: Não é necessário. Basta analisar o posto da matriz A. Há três possibilidades:

i. Se $rank(A) < rank([A|b])$, então não existe solução para o sistema de equações lineares e, portanto, o problema é infactível.

ii. Se $rank(A) = rank([A|b]) = n$, então o sistema de equações lineares tem solução única, podendo ser obtida através de métodos de solução de equações lineares como, por exemplo, método de Gauss e fatoração SVD.

iii. Se $rank(A) = rank([A|b]) < n$, então existem infinitas soluções para o sistema de equações lineares e faz sentido realizar otimização. Neste caso, a análise deve ser refinada. Seja $\bar{x}$ uma solução para $Ax = b$ e seja N uma base para o espaço nulo de A, ou seja, $null(A) = \{Ny : y \in \mathbb{R}^{n-rank(A)}\}$. Podemos reescrever o problema como segue:

$$\begin{array}{lll}
\text{Min} \quad c^T x & & \text{Min} \quad c^T(\bar{x} + Ny) \\
\text{S.a:} \quad Ax = b & \Leftrightarrow & \text{S.a:} \quad A(\bar{x} + Ny) = b \\
& \Leftrightarrow & \text{Min} \quad c^T\bar{x} + c^T Ny \\
& & \text{S.a:} \quad A\bar{x} + ANy = b \\
& \Leftrightarrow & \text{Min} \quad c^T Ny \\
& & \text{S.a:} \quad b + 0 = b \\
& \Leftrightarrow & \text{Min} \quad c^T Ny
\end{array}$$

Logo o problema tem um número infinito de soluções ótimas, quando $c^T N = 0$, ou o problema é ilimitado caso $c^T N \neq 0$.

Assim, o problema pode ser resolvido facilmente considerando os três casos acima.

EX 6.10: Duas formulações em programação linear, P_1 e P_2, são dadas para o pro-

blema P. Segue a formulação P_1:

$$\begin{array}{rll} P_1: & \text{Min} & \sum_{j=1}^{m} d_j \\ & \text{S.a}: & \\ & & -d_j \leq a_j^T x - b_j \leq d_j,\ j = 1 \ldots, m \\ & & d_j \geq 0,\ j = 1, \ldots, m \\ & & l \leq x \leq u \end{array}$$

sendo $A = \begin{bmatrix} a_1^T \\ a_2^T \\ \vdots \\ a_m^T \end{bmatrix}$ e $a_j \in \mathbb{R}^{n \times 1}$, $j = 1 \ldots, m$.

Segue a formulação P_2:

$$\begin{array}{rll} P_2: & \text{Min} & \sum_{j=1}^{m} (e_j^+ + e_j^-) \\ & \text{S.a}: & \\ & & a_j^T x - b_j = e_j^+ - e_j^-,\ j = 1 \ldots, m \\ & & e_j^+, e_j^- \geq 0,\ j = 1, \ldots, m \\ & & l \leq x \leq u \end{array}$$

A.7 Teoria dos Jogos

EX 7.1: Aplicando o formalismo apresentado, pode-se verificar que a estratégia ótima do agente coluna é

$$x_1 = 1/4,\ x_2 = 3/4,\ \text{e}\ x_3 = 0$$

enquanto que a estratégia ótima do agente linha é

$$y_1 = 3/4,\ y_2 = 1/4,\ \text{e}\ y_3 = 0$$

O jogo não é justo tendo valor 0.75, o que indica uma ligeira vantagem para o agente coluna.

EX 7.2: Seja A o jogador "linha" e B o jogador "coluna". A matriz do jogo pode ser expressa como:

$$A = \begin{bmatrix} -5 & 5 \\ 10 & -10 \end{bmatrix}$$

i. Quais são as estratégias ótimas?
Seja y_5 (y_{10}) a probabilidade do jogador A esconder a moeda de 5 centavos (10 centavos). Seja x_5 (x_{10}) a probabilidade do jogador B esconder a moeda de 5 centavos (10 centavos). Resolvendo o problema, pode-se verificar que:

$$x^* = (x_5^*, x_{10}^*) = (1/2, 1/2)$$

$$y^* = (y_5^*, y_{10}^*) = (2/3, 1/3)$$

ii. Qual dos jogadores tem vantagem?
Nenhum dos jogadores tem vantagem, pois o valor do jogo é $z^* = w^* = 0$.

iii. Resolva o problema para quaisquer denominações.
Tomemos primeiramente a matriz A do jogo como sendo

$$A = \begin{bmatrix} -a & a \\ b & -b \end{bmatrix}$$

O problema do agente coluna (x) pode ser colocado nas formas equivalentes:

$$\begin{array}{lll} P_x: & \text{Max} & v \\ & \text{S.a:} & \\ & & \begin{array}{rcrcrcl} ax_a & - & ax_b & + & v & \leq & 0 \\ -bx_a & + & bx_b & + & v & \leq & 0 \\ x_a & + & x_b & & & = & 1 \end{array} \\ & & x_a, x_b \geq 0 \end{array}$$

$$\begin{array}{lll} P_x: & \text{Max} & v \\ & \text{S.a:} & \\ & & \begin{array}{rcrcrcrcrcl} ax_a & - & ax_b & + & v & + & w_1 & & & = & 0 \\ -bx_a & + & bx_b & + & v & + & & + & w_2 & = & 0 \\ x_a & + & x_b & & & & & & & = & 1 \end{array} \\ & & x_a, x_b, w_1, w_2 \geq 0 \end{array}$$

O problema pode ser resolvido por meio dos passos a seguir.

1) $v = -ax_a + ax_b - w_1$

2)

$$\begin{aligned} bx_b &= bx_a - v - w_2 \\ &= bx_a + ax_a - ax_b + w_1 - w_2 \\ &= (a+b)x_a - ax_b + w_1 - w_2 \Rightarrow \\ (a+b)x_b &= (a+b)x_a + w_1 - w_2 \Rightarrow \\ x_b &= x_a + (w_1 - w_2)/(a+b) \end{aligned}$$

3)

$$\begin{aligned} x_a + x_b = 1 &\Rightarrow x_a + (x_a + (w_1 - w_2)/(a+b)) = 1 \\ &\Rightarrow 2x_a = 1 + (w_2 - w_1)/(a+b) \\ &\Rightarrow x_a = \frac{1}{2} + \frac{w_2 - w_1}{2(a+b)} \end{aligned}$$

4)

$$\begin{aligned} x_b &= x_a + (w_1 - w_2)/(a+b) \\ &= \frac{1}{2} + \frac{w_2 - w_1}{2(a+b)} + \frac{w_1 - w_2}{a+b} \\ &= \frac{1}{2} - \frac{w_1 - w_2}{2(a+b)} + 2\frac{w_1 - w_2}{2(a+b)} \\ &= \frac{1}{2} + \frac{w_1 - w_2}{2(a+b)} \end{aligned}$$

5) Podemos deduzir a partir dos itens (1), (3) e (4) que

$$\begin{aligned} v &= -ax_a + ax_b - w_1 \\ &= a(x_b - x_a) - w_1 \\ &= a(\frac{1}{2} + \frac{w_1 - w_2}{2(a+b)} - \frac{1}{2} - \frac{w_2 - w_1}{2(a+b)}) - w_1 \\ &= a\frac{(w_1 - w_2)}{(a+b)} - w_1 \\ &= -\frac{(bw_1 + aw_2)}{a+b} \end{aligned}$$

6) Fazendo $w_1 = 0$ e $w_2 = 0$ variáveis não básicas, podemos concluir a partir do desenvolvimento acima que $v \geq 0$. Tomando o dicionário:

$$\begin{aligned} \delta &= 0 - \tfrac{bw_1}{a+b} + \tfrac{aw_2}{a+b} \\ x_a &= \tfrac{1}{2} - \tfrac{w_1}{2(a+b)} + \tfrac{w_2}{2(a+b)} \\ x_b &= \tfrac{1}{2} + \tfrac{w_1}{2(a+b)} - \tfrac{w_2}{2(a+b)} \\ v &= 0 - \tfrac{bw_1}{a+b} + \tfrac{aw_2}{a+b} \end{aligned}$$

observamos que $\delta = v$ (dependência linear da equação objetivo e da última linha) e a solução ótima difere apenas de qual variável é básica (v, w_1 ou w_2). Logo, $x_a = \frac{1}{2}$ e $x_b = \frac{1}{2}$. Assim, a estratégia ótima do agente coluna (x) é $x_a^* = \frac{1}{2}$ e $x_b^* = \frac{1}{2}$, induzindo o jogo a ter valor $v^* = 0$.

7) Para obtermos a estratégia ótima do jogador linha (y), vamos tomar o dual de P_x:

$$
\begin{array}{lll}
P_y: & \text{Min} & u \\
& \text{S.a:} & \\
& & \begin{array}{rcrcrcl}
ay_a & - & by_b & + & u & \geq & 0 \\
-ay_a & + & by_b & + & u & \geq & 0 \\
y_a & + & y_b & & & = & 1 \\
y_a, y_b \geq 0 & & & & & &
\end{array}
\end{array}
$$

Uma vez que $v^* = 0$, segue que $u^* = 0$. Observando que $x_a^* > 0$ e $x_b^* > 0$, temos pelo Teorema da Folga Complementar que:

$$
\left\{
\begin{array}{rcrcl}
ay_a & - & by_b & = & 0 \\
-ay_a & + & by_b & = & 0 \\
y_a & + & y_b & = & 1
\end{array}
\right. \Rightarrow y_b = 1 - y_a.
$$

Logo, podemos concluir que $y_a = \frac{b}{a+b}$ e $y_b = \frac{a}{a+b}$. Em síntese, as estratégias ótimas para denominações a e b são:

$$
\left\{
\begin{array}{rcl}
x_a^* & = & \frac{1}{2} \\
x_b^* & = & \frac{1}{2} \\
y_a^* & = & \frac{b}{a+b} \\
y_b^* & = & \frac{a}{a+b}
\end{array}
\right.
$$

A.8 Fluxo em Redes

EX 8.1: O problema de fluxo em redes de custo mínimo (primal) pode ser especificado como segue para uma rede $G = (N, A)$:

$$
\begin{aligned}
P: \quad & \text{Minimize} \quad \sum_{(i,j)\in A} c_{ij}x_{ij} \\
& \text{Sujeito a:} \\
& \qquad \sum_{\{j:(i,j)\in A\}} x_{ij} - \sum_{\{j:(j,i)\in A\}} x_{ji} = b(i) \qquad i \in N \\
& \qquad x_{ij} \leq u_{ij} \qquad (i,j) \in A \\
& \qquad -x_{ij} \leq -l_{ij} \qquad (i,j) \in A
\end{aligned}
$$

O dual de P pode ser expresso como:

$$
\begin{aligned}
D: \quad & \text{Maximize} \quad \sum_{i\in N} b(i)y_i + \sum_{(i,j)\in A} u_{ij}w_{ij} - \sum_{(i,j)\in A} l_{ij}v_{ij} \\
& \text{Sujeito a:} \\
& \qquad y_i - y_j + w_{ij} - v_{ij} \leq c_{ij} \qquad (i,j) \in A \\
& \qquad y_i \in \mathbb{R} \qquad i \in A \\
& \qquad w_{ij}, v_{ij} \geq 0 \qquad (i,j) \in A
\end{aligned}
$$

EX 8.2: a) Formule o problema de caminhos mínimos de um vértice s para um vértice t como um problema de fluxo em redes. Seja $G = (N, A)$ um grafo direcionado e w_{ij} o peso da aresta $(i, j) \in A$. Defina a capacidade de cada arco como a unidade: $u_{ij} = 1,\ \forall (i, j) \in A$. Defina o limite inferior para o fluxo de cada arco: $l_{ij} = 0,\ \forall (i, j) \in A$. Defina o custo de cada arco como o seu peso: $c_{ij} = w_{ij},\ \forall (i, j) \in A$. Defina o fluxo injetado em cada nó: $b(s) = 1$, $b(t) = -1$ e $b(i) = 0,\ \forall i \in N - \{s, t\}$. O fluxo ótimo em G é precisamente um caminho mínimo de s para t.

b) Formule o problema de encontrar a árvore de caminhos mínimos a partir de um vértice s para os demais vértices do grafo. Seja $G = (N, A)$ um grafo direcionado e w_{ij} o peso da aresta $(i, j) \in A$. Defina a capacidade de cada arco como a unidade: $u_{ij} = |N| - 1,\ \forall (i, j) \in A$. Defina o limite inferior para o fluxo de cada arco: $l_{ij} = 0,\ \forall (i, j) \in A$. Defina o custo de cada arco como o seu peso: $c_{ij} = w_{ij},\ \forall (i, j) \in A$. Defina o fluxo injetado em cada nó: $b(s) = |N| - 1$ e $b(i) = -1,\ \forall i \in N - \{s\}$. O fluxo ótimo em G induz uma árvore de caminhos mínimos de s para t.

EX 8.3: A solução ótima é dada por:

$$x_{14} = 10,\ x_{21} = 0,\ x_{23} = 10, x_{26} = 10,\ x_{34} = 5$$

$$x_{35} = 5,\ x_{47} = 10,\ x_{56} = 0,\ x_{57} = 5,\ x_{68} = 10$$

$$x_{78} = 0$$

O fluxo ótimo correspondente tem custo 270.

EX 8.4: *A afirmação é verdadeira.*

Primeiro, pode-se mostrar que existe uma solução factível cujos fluxos através dos arcos são todos pares. Para obter-se uma solução factível, utilize o modelo de fluxo máximo em redes construindo uma rede $G' = (N', A')$ a partir de G ao eliminar-se os custos, c_{ij}, e os parâmetros de suprimento/demanda, $b(i)$, introduzindo um nó fonte s e os arcos (s, i) com capacidade $b(i)$ para todo $i \in N$ tal que $b(i) > 0$, e introduzindo um nó terminal t e os arcos (i, t) com capacidade $-b(i)$ para todo $i \in N$ tal que $b(i) < 0$. É fácil de se convencer que G possui uma solução factível se, e somente se, o fluxo máximo de s para t em G' é igual a $\sum_{\{i \in N: b(i) > 0\}} b(i) = -\sum_{\{i \in N: b(i) < 0\}} b(i)$. O algoritmo de aumento para fluxo máximo iterativamente encontra um caminho com capacidade positiva, conectando s a t, e enviando o fluxo máximo permitido até que não existam caminhos restantes. É fácil de se verificar a invariância de que o fluxo é sempre par, dado que as capacidades dos arcos e os valores de suprimento/demanda sejam todos pares. Portanto, a solução ótima obtida induz valores pares para todos os fluxos.

Segundo, basta se convencer de que o algoritmo de cancelamento de circuito negativo, a cada iteração, obtém uma solução que induz valores pares para os fluxos de todos os arcos.

EX 8.5: *O conjunto de soluções ótimas para um problema qualquer de fluxo em redes, de custo mínimo, se altera se multiplicarmos o custo de cada arco por uma constante k?*

Se multiplicarmos os custos por uma constante $k \leq 0$, então o conjunto de soluções ótimas é tipicamente alterado. Se multiplicarmos os custos por uma constante $k > 0$, então a partir da linearidade da função objetivo conluí-se que o conjunto de soluções ótimas não se altera.

O conjunto se altera se adicionarmos uma constante k?

Dependendo da constante k, o conjunto de soluções ótimas pode ser alterado ou não. Para a rede da Figura A.2, o conjunto de soluções ótimas não se altera se $k = 1$, mas é modificado quando $k = 100$.

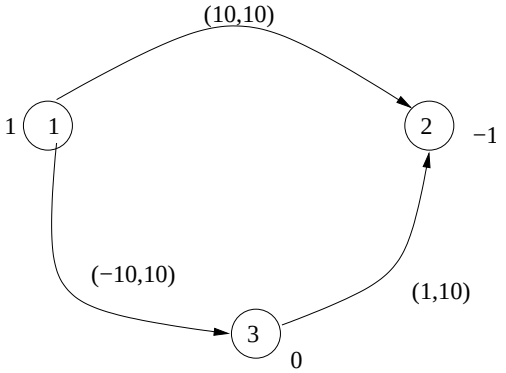


Figura A.2: Exemplo de problema de fluxo em redes

EX 8.6: Substitua cada vértice i pelos vértices i_1 e i_2. Substitua cada arco (i, j) por (i_2, j_1), utilizando a mesma capacidade e custo do arco original. Para cada vértice i, introduza o arco (i_1, i_2) definindo $c(i_1, i_2) = 0$ e $u(i_1, i_2) = w(i)$. Para cada vértice i com $b(i) \geq 0$, faça $b(i_1) = 0$ e $b(i_2) = b(i)$. Para cada vértice i com $b(i) < 0$, faça $b(i_1) = b(i)$ e $b(i_2) = 0$.

EX 8.7: Ver exercício EX 8.4.

EX 8.8: 1) mostre como formular qualquer problema de fluxo em redes de custo mínimo como um problema custo-mínimo fluxo-máximo.

Seja $N_+ = \{i \in N : b(i) > 0\}$, $N_0 = \{i \in N : b(i) = 0\}$ e $N_- = \{i \in N : b(i) < 0\}$. Aumente a rede G introduzindo os nós s e t. Adicione os arcos (si) para cada $i \in N_+$, fazendo $c_{si} = 0$ e $u_{si} = b(i)$, e (i, t) para cada $i \in N_-$, fazendo $c_{it} = 0$ e $u_{it} = -b(i)$. Faça $b(s) = \sum_{i \in N_+} b(i)$, $b(t) = \sum_{i \in N_-} b(i)$ e $b(i) = 0$ para todo $i \in N_+ \cup N_-$. Então a solução do problema de fluxo-máximo custo-mínimo na rede aumentada produz uma solução ótima para o problema de fluxo em rede de custo mínimo na rede original.

2) mostre como o problema de custo-mínimo fluxo-máximo pode ser resolvido por meio de uma redução ao problema de custo mínimo.

Modifique a rede introduzindo um arco (t, s) sendo $u_{ts} = \infty$ e $c_{ts} = -\sum_{(i,j) \in A} |c_{ij}| - k$, para $k > 0$. Faça $b(i) = 0$ para todo $i \in N$. O fluxo de custo mínimo na rede aumentada produz um fluxo-máximo de custo mínimo na rede original.

EX 8.9: Primeiramente, obtemos a rede $G' = (V', A')$ a partir da operação de quebra de vértices do grafo G: $V' = \{i, i' : i \in V\}$ e $A' = \{(i, i') : i \in V\} \cup \{(i', j) : (i, j) \in A\}$. Defina os seguintes custos e capacidades: $c_{i',j} = c_{i,j}$ para todo $(i, j) \in A$, $c_{i,i'} = p_i$ para todo $i \in V$, e $u_{i,j} = 1$ para todo $(i, j) \in A'$. Defina ainda $b_i = 0$ para todo $i \in V'$, exceto $b_s = 1$ e $b_{t'} = 1$. Então o caminho com menor custo de translado e pedágio pode ser encontrado com o algoritmo de fluxo em redes de custo mínimo. Como exemplo, considere o problema de identificar um caminho de menor custo de $s = 1$ para $t = 4$ conforme grafo da Figura A.3.

EX 8.10: Começamos gerando o grafo direcionado $G = (V, A)$ tal que:

a) a capacidade de cada arco $(i, j) \in A$ é $u_{ij} = x_{ij}$, onde x_{ij} é o valor gerado pela solução candidata x;

b) $l_{ij} = 0$ para todo $(i, j) \in A$.

Desejamos encontrar um corte $[S, \bar{S}]$ com capacidade $u[S, \bar{S}]$ mínima em G. Se $u[S, \bar{S}] = \sum_{\{(i,j) \in A : i \in S, j \in V - S\}} u_{ij} < 1$, então a solução candidata não satisfaz a restrição de eliminação de sub-rota para S. Portanto, o problema em questão se reduz a encontrar um corte mínimo em G conforme definição acima. Isto pode ser realizado da seguinte forma. Fixe um vértice qualquer $s \in V$. O corte mínimo deve ter $s \in S$ e um vértice $t \in V - S$. Como não conhecemos o vértice t, resolvemos $|V| - 1$ problemas de fluxo máximo entre s e t, um para cada $t \in V - \{s\}$, obtendo o corte mínimo $[S, \bar{S}]$. O corte mínimo é o menor dentre

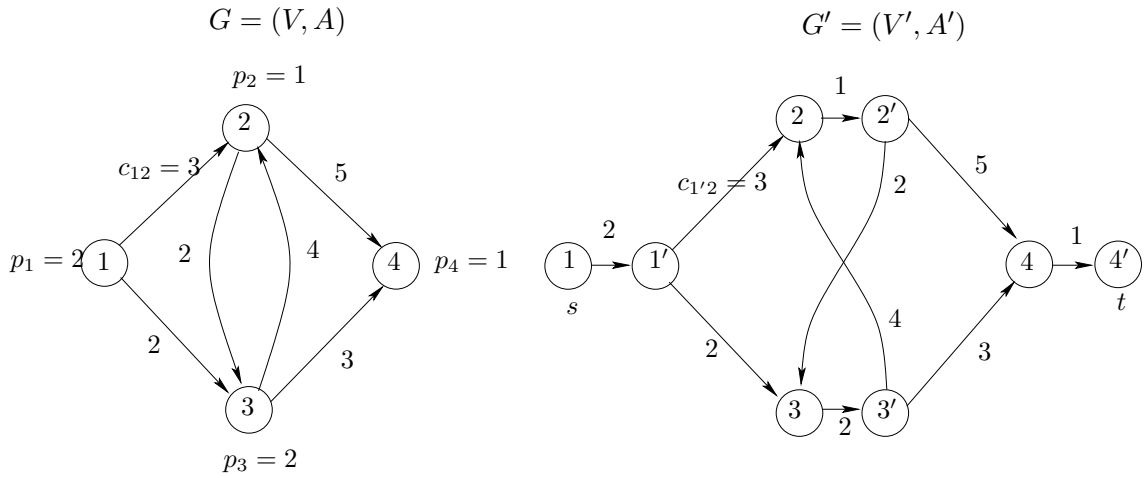


Figura A.3: Ilustração da redução do problema de caminhos mínimos com pedágio ao problema de fluxo de custo mínimo através da operação de quebra de vértices.

todos estes cortes. Logo, o problema pode ser resolvido computando $|V|-1$ fluxos máximos em G. Isto decorre do teorema fluxo-máximo corte-mínimo.

EX 8.11: O dual do problema de fluxo máximo pode ser expresso como:

$$
\begin{array}{lll}
D: & \text{Minimize} & \sum_{(i,j)\in E} u_{ij} w_{ij} \\
& \text{Sujeito a :} & \\
& & y_i - y_j + w_{ij} \geq 0, \quad \forall (i,j) \in A \\
& & -y_s + y_t \geq 1 \\
& & w_{ij} \geq 0, \qquad \forall (i,j) \in A \\
& & y_i \in \mathbb{R}, \qquad \forall i \in V
\end{array}
$$

A.9 Linguagens de Modelagem

EX 9.1: O problema pode ser colocado em programação matemática :

$$\begin{array}{ll} \text{Minimize} & \sum_{j=1}^{n} s_j w_j \\ \text{Sujeito a :} & \\ & s_j \geq s_i + p_i - M(1 - x_{ij}) \qquad i = 1, \ldots, n \\ & \qquad\qquad\qquad\qquad\qquad\quad j = 1, \ldots, n, j \neq i \\ & x_{ij} + x_{ji} = 1 \qquad\qquad\quad i = 1, \ldots, n \\ & \qquad\qquad\qquad\qquad\qquad\quad j = i+1, \ldots, n \\ & s_j \geq 0 \qquad\qquad\qquad\quad j = 1, \ldots, n \\ & x_{ij} \in \{0, 1\} \qquad\qquad\quad i = 1, \ldots, n \\ & \qquad\qquad\qquad\qquad\qquad\quad j = i = 1, \ldots, n \\ & M = \sum_{j=1}^{n} p_j \end{array}$$

A solução ótima sequencia as tarefas na ordem: $8 \rightarrow 4 \rightarrow 9 \rightarrow 1 \rightarrow 5 \rightarrow 10 \rightarrow 2 \rightarrow 6 \rightarrow 3 \rightarrow 7$. O custo desta solução é: 24.69. Em AMPL, o modelo pode ser expresso como:

```
#-------------------------------------------------------
# WellModel1.mod
#

param n > 0 integer;

param w     {j in 1..n} >= 0;
param p     {j in 1..n} >= 0;
param M >= 0;


#-------------------------------------------------------------
var x   { i in 1..n, j in 1..n : i!=j } integer >=0, <=1;
var s   { j in 1..n} >= 0;


#-------------------------------------------------------------
minimize cost:
      sum{j in 1..n} w[j]*s[j];


#-------------------------------------------------------------
subject to prec1{i in 1..n, j in 1..n : i!=j}:
     s[j] >= s[i] + p[i] - M*(1 - x[i,j]);

subject to prec2{i in 1..n, j in (i+1)..n }:
      x[i,j] + x[j,i] = 1;
```

O arquivo de dados em AMPL tem como conteúdo:

```
param n := 10;

param M := 25;

param:  w   p :=
1   0.5 1.5
2   0.4 2.3
3   0.1 1.2
4   0.5 1.0
5   0.3 1.5
6   0.5 3.6
7   0.2 4.1
8   0.8 0.8
9   1.0 2.0
10  0.6 3.0;
```

Em Mosel (Xpress-MP), o problema pode ser especificado conforme o modelo:

```
model scheduling

uses "mmxprs"

declarations

  Jobs: set of integer
  M : real
  w : array(Jobs) of real
  p : array(Jobs) of real
  x: array(Jobs,Jobs) of mpvar
  s: array(Jobs) of mpvar

end-declarations

!Jobs:= {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

!M:= 25

!w:= [0.5 0.4 0.1 0.5 0.3 0.5 0.2 0.8 1.0 0.6]
!p:= [1.5 2.3 1.2 1.0 1.5 3.6 4.1 0.8 2.0 3.0]

initializations from "model.dat"
  Jobs M w p
end-initializations

forall(i in Jobs, j in Jobs | i<>j )
  create(x(i,j))
```

```
forall(i in Jobs, j in Jobs | i<>j )
  x(i,j) is_binary

forall(j in Jobs)
  create(s(j))

! Constraints
forall(j in Jobs)
  s(j) >= 0

! Constraints
forall(i in Jobs, j in Jobs | i < j)
  x(i,j) + x(j,i) = 1

forall(i in Jobs, j in Jobs | i<>j )
  s(j) >= s(i) + p(i) -M*(1 - x(i,j))

! Objective Function
Cost := sum(j in Jobs)
    s(j)*w(j)

minimize(Cost)

writeln
writeln("-------------------------------------")
forall(i in Jobs, j in Jobs | i<>j )
  writeln("x(",i,",",j,"): ",getsol(x(i,j)))

writeln("-------------------------------------")
forall(j in Jobs)
  writeln("s(",j,"): ",getsol(s(j)))

writeln
writeln("objective: ",getobjval)
```

O conteúdo do arquivo de dados definindo a instância é:

```
Jobs: [1 2 3 4 5 6 7 8 9 10]

M: 25

w: [0.5 0.4 0.1 0.5 0.3 0.5 0.2 0.8 1.0 0.6]
p: [1.5 2.3 1.2 1.0 1.5 3.6 4.1 0.8 2.0 3.0]
```

A.10 Programação Inteira: Fundamentos

EX 10.1: Seja $x_j = 1$ se o investimento $j \in S$ é escolhido e, caso contrário, $x_j = 0$.

i. Não pode-se investir em todas as ações: $\sum_{j=1}^{7} x_j \leqslant 6$.

ii. Pelo menos uma ação deve ser selecionada: $\sum_{j=1}^{7} x_j \geqslant 1$.

iii. A ação 1 não pode ser escolhida se a ação 3 é escolhida: $x_1 + x_3 \leqslant 1$.

iv. A ação 4 pode ser escolhida apenas se a ação 2 é escolhida: $x_4 \leqslant x_2$.

v. Ambas as ações 1 e 5 devem ser escolhidas ou, exclusivamente, nenhuma delas: $x_1 = x_5$.

vi. Pelo menos uma das ações do conjunto $\{1, 2, 3\}$ ou pelo menos duas ações do conjunto $\{2, 4, 5, 6\}$:

$$2(x_1+x_2+x_3)+(x_2+x_4+x_5+x_6) \geqslant 2 \Rightarrow 2(x_1+x_2+x_3)+(x_4+x_5+x_6) \geqslant 2$$

EX 10.2: i. $u = \min\{x_1, x_2\}$, assumindo que $0 \leqslant x_j \leqslant C$ para $j = 1, 2$:

$$\begin{aligned} u &\leqslant x_1 \\ u &\leqslant x_2 \\ u &\geqslant x_1 - Cy \\ u &\geqslant x_2 - C(1-y) \end{aligned}$$

onde

$$\begin{aligned} &y \in \{0, 1\} \\ &0 \leqslant x_j \leqslant C, \ j = 1, 2. \end{aligned}$$

ii. $v = |x_1 - x_2|$ com $0 \leqslant x_j \leqslant C$ para $j = 1, 2$:

$$\begin{aligned} v &\geqslant x_1 - x_2 \\ v &\geqslant x_2 - x_1 \\ v &\leqslant x_1 - x_2 + 2Cy \\ v &\leqslant x_2 - x_1 + 2C(1-y) \end{aligned}$$

onde

$$\begin{aligned} &y \in \{0, 1\} \\ &0 \leqslant x_j \leqslant C, \ j = 1, 2. \end{aligned}$$

iii. O conjunto $X - \{x^*\}$ onde $X = \{x \in Z^n : Ax \leqslant b\}$ e $x^* \in X$.

$$\begin{aligned} X - \{x^*\} = \{x \in Z^n : \ & Ax \leqslant b \\ & x^T x^* \leqslant e^T x^* + n(1-y) \\ & x^T(e - x^*) \geqslant 1 - y \\ & y \in \{0, 1\} \qquad\qquad \} \end{aligned}$$

EX 10.3: Basta verificar que

$$\begin{aligned} X_1 = X_2 = X_3 = \{ \ & (0,0,0,0), \ (0,0,0,1), \ (0,0,1,0), \ (0,0,1,1), \\ & (0,1,0,0), \ (0,1,0,1), \ (0,1,1,0), \ (0,1,1,1), \\ & (1,0,0,0), \ (1,0,0,1), \ (1,0,1,0), \ (1,1,0,0)\} \end{aligned}$$

EX 10.4: Solução ótima: Rota $1-2-4-5-6-8-7-3-1$ com comprimento 13. O código LP-SOLVE segue abaixo.

```
min: x12 + x13 + x23 + 2*x24 + 5*x26 + 2*x34 + 4*x37 +
     x45 + 2*x56 + 2*x57 + x68 + x78;

x12 + x13 = 2;
x12 + x23 + x24 + x26 = 2;
x13 + x23 + x34 + x37 = 2;
x24 + x34 + x45 = 2;
x45 + x56 + x57 = 2;
x26 + x56 + x68 = 2;
x37 + x57 + x78 = 2;
x68 + x78 = 2;

x26 + x45 + x37 >= 2;

x12 <= 1;
x13 <= 1;
x23 <= 1;
x24 <= 1;
x26 <= 1;
x34 <= 1;
x37 <= 1;
x45 <= 1;
x56 <= 1;
x57 <= 1;
x68 <= 1;
x78 <= 1;


int x12;
int x13;
int x23;
int x24;
int x26;
int x34;
int x37;
int x45;
int x56;
int x57;
int x68;
int x78;
```

EX 10.5: 1) y_{ij} é igual a $x_i x_j$:

$$\begin{cases} y_{ij} & \geqslant & x_i + x_j - 1 \\ y_{ij} & \leqslant & x_i \\ y_{ij} & \leqslant & x_j \\ y_{ij} & \geqslant & 0 \end{cases} \qquad \text{ou } y_{ij} + 1 \geqslant x_i + x_j \geqslant 2y_{ij}$$

2) Se $x_1 = 1$ então $x_2 + x_3 = 1$:

$$\begin{cases} x_2 + x_3 & \leqslant & 2 - x_1 \\ x_2 + x_3 & \geqslant & x_1 \end{cases}$$

3) x_3 é 1 se e somente se x_1 é 1 ou x_2 é 1, mas não ambas:

$$\begin{cases} x_3 & \geqslant & x_1 - x_2 \\ x_3 & \geqslant & x_2 - x_1 \\ x_3 & \leqslant & 2 - x_1 - x_2 \\ x_3 & \leqslant & x_1 + x_2 \end{cases}$$

4) x_1 e x_2 podem assumir todos os valores 0-1 possíveis, exceto $x_1 = 0$ e $x_2 = 1$:

$$x_1 \geqslant x_2$$

EX 10.6: As respostas seguem abaixo.

i. As variáveis x_j podem ser formuladas como segue:

$$\begin{cases} x_0 = 0 \\ x_j \leqslant x_{j-1} + a_{j-1} \\ x_j \geqslant x_{j-1} + a_{j-1} - n(1 - a_{j-1}) \\ x_j \geqslant 0 \\ x_j \leqslant n a_{j-1} \end{cases}, \; j = 1, \ldots, n$$

ii. O limite máximo para x_j pode ser modelado por:

$$x_j \leqslant l, \; j = 1, \ldots, n$$

iii. O comprimento mínimo de subsequências de 1's pode ser modelado por:

$$a_j \geqslant \frac{l_{min} - x_{j-1} - l_{min} n(1 - a_{j-1})}{l_{min}}, \; j = 1, \ldots, n$$

mas permite que se tenha uma subsequência de comprimento menor que l_{min} no extremo direito da sequência a.

EX 10.7: Seja $a^n \in \mathbb{B}^{M \times 1}$ um vetor tal que $a_m^n = 1$ se a rota r_n passa por p_m e $a_m^n = 0$ caso contrário, para $n = 1, \ldots, N$ e $m = 1, \ldots, M$. O problema pode ser colocado como um problema de cobertura de conjuntos:

$$\begin{array}{ll} \text{Minimize} & \sum_{n=1}^{N} c_n x_n \\ \text{Sujeito a :} & \\ & \sum_{n=1}^{N} a_m^n \geqslant 1, \; m = 1, \ldots, M \\ & x_n \in \{0, 1\}, \; n = 1, \ldots, N \end{array}$$

EX 10.8: O problema pode ser colocado formalmente como:

$$
\begin{array}{lll}
P: & \text{Minimize} & \sum_{j=1}^{n} d_j \\
& \text{Sujeito a :} & \\
& & \left\{ \begin{array}{ll} -\hat{x}_j \leqslant x_j - x \leqslant \hat{x}_j & \\ -\hat{y}_j \leqslant y_j - y \leqslant \hat{y}_j & \\ \hat{x}_j \geqslant 0 & , \quad j = 1, \ldots, n \\ \hat{y}_j \geqslant 0 & \\ d_j = \min\{\hat{x}_j, \hat{y}_j\} & \end{array} \right.
\end{array}
$$

Para escrever a função min em programação matemática, podemos fazer uso do exercício já resolvido anteriormente. Logo, obtemos uma formulação do problema em programação inteira mista:

$$
\begin{array}{lll}
P_I: & \text{Minimize} & \sum_{j=1}^{n} d_j \\
& \text{Sujeito a :} & \\
& & \left\{ \begin{array}{rcll} -\hat{x}_j & \leqslant & x_j - x \leqslant \hat{x}_j & \\ -\hat{y}_j & \leqslant & y_j - y \leqslant \hat{y}_j & \\ \hat{x}_j & \geqslant & 0 & \\ \hat{y}_j & \geqslant & 0 & \\ d_j & \leqslant & \hat{x}_j & , \quad j = 1, \ldots, n \\ d_j & \leqslant & \hat{y}_j & \\ d_j & \geqslant & \hat{x}_j - Mz_j & \\ d_j & \geqslant & \hat{y}_j - M(1 - z_j) & \\ & & z_j \in \{0, 1\} & \end{array} \right.
\end{array}
$$

onde $|x_j - x| \leqslant \hat{x}_j$, $|y_j - y| \leqslant \hat{y}_j$, $d_j = \min\{\hat{x}_j, \hat{y}_j\}$, $\hat{x}_j = \min\{\hat{x}_j, \hat{y}_j\}$ se $z_j = 0$ e $\hat{y}_j = \min\{\hat{x}_j, \hat{y}_j\}$ se $z_j = 1$, onde M é uma constante suficientemente grande. Note que para uma solução ser ótima, $|x_j - x| = \hat{x}_j$ se $z_j = 0$ e $|y_j - y| = \hat{y}_j$ se $z_j = 1$.

O problema P' pode ser colocado em uma forma alternativa, com um número maior de variáveis mas um número menor de restrições:

$$
\begin{array}{lll}
P': & \text{Minimize} & \sum_{j=1}^{n} d_j \\
& \text{Sujeito a :} & \\
& & \left\{ \begin{array}{ll} x_j - x = x_j^+ - x_j^- & \\ y_j - y = y_j^+ - y_j^- & \\ x_j^+, x_j^- \geqslant 0 & , \quad j = 1, \ldots, n \\ y_j^+, y_j^- \geqslant 0 & \\ d_j = \min\{x_j^+ + x_j^-, y_j^+ + y_j^-\} & \end{array} \right.
\end{array}
$$

Escrevendo o operador min em termos de restrições lineares, podemos reexpressar

P' com um problema de programação inteira mista:

$$
\begin{aligned}
P'_I: \quad & \text{Minimize} \quad \sum_{j=1}^{n} d_j \\
& \text{Sujeito a :} \\
& \begin{cases}
x_j - x = x_j^+ - x_j^- \\
y_j - y = y_j^+ - y_j^- \\
x_j^+, x_j^- \geqslant 0 \\
y_j^+, y_j^- \geqslant 0 \\
d_j \leqslant x_j^+ + x_j^- \\
d_j \leqslant y_j^+ + y_j^- \\
d_j \geqslant x_j^+ + x_j^- - Mz_j \\
d_j \geqslant y_j^+ + y_j^- - M(1 - z_j) \\
z_j \in \{0, 1\}
\end{cases} , \quad j = 1, \ldots, n
\end{aligned}
$$

Resolvendo o modelo P_I para a instância dada no exercício, obtemos a solução $(x, y) = (383, 442)$ com custo 547. A solução obtida está ilustrada na Figura A.4.

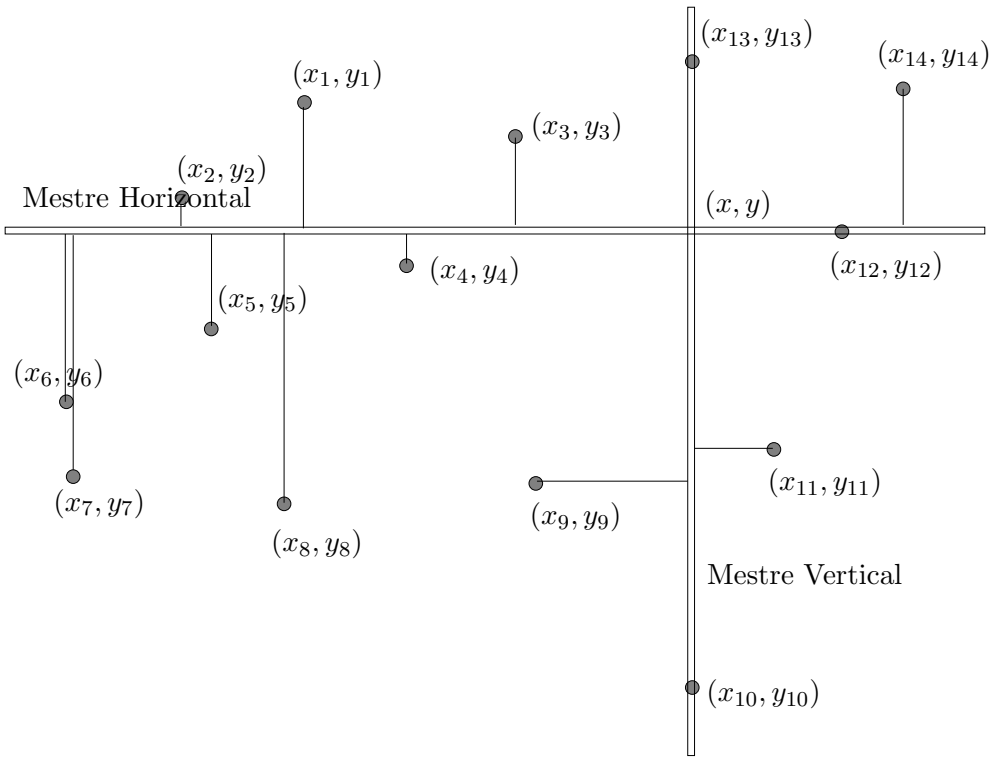


Figura A.4: Rede de abastecimento.

A.11 Programação Inteira: Relaxações e Algoritmo *Branch-and-Bound*

EX 11.1: Seja P o problema original dado por:

$$
\begin{array}{lll}
P: \text{ Maximize } & 77.9x_1 + 76.8x_2 + 89.6x_3 + 97.1y_1 + 31.3y_2 & = \quad x_0 \\
\text{Sujeito a :} & & \\
& 60.9x_1 + 68.9x_2 + 69.0x_3 - 56.9y_1 + 22.5y_2 & = \quad 86.5 \\
& -86.8x_1 + 32.7x_2 + 24.3x_3 + 13.8y_1 - 12.6y_2 & \leq \quad 77.3 \\
& 10.9x_1 + 3.6x_2 - 40.8x_3 + 43.9y_1 + 7.1y_2 & \leq \quad 82.3 \\
& x_1, x_2, x_3 \geq 0 \text{ e inteiro} & \\
& y_1, y_2 \geq 0 &
\end{array}
$$

(*Bounding*) Resolvendo a relaxação $R(P)$ obtemos a solução $x_1 = 1.496$, $x_2 = 0$, $x_3 = 5.021$, $y_1 = 6.1697$ e $y_2 = 0$ e o limite superior 1165 para o valor objetivo da solução ótima.

(*Branching*) Quebrando o problema em dois subproblemas $P_1 = P \cap \{x : x_1 \leq 1\}$ e $P_2 = P \cap \{x :\geq 2\}$, obtemos a árvore *branch-and-bound* da Figura A.5.

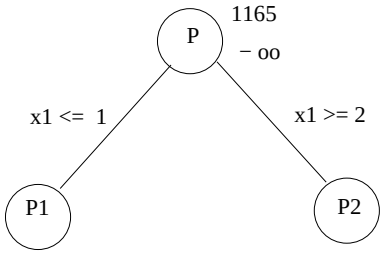


Figura A.5: Árvore *branch-and-bound* parcial

(*Bounding*) Resolvendo a relaxação $R(P_2)$ do problema P_2, obtemos a solução $x_1 = 2$, $x_2 = 0$, $x_3 = 2.675$, $y_1 = 3.8642$ e $y_2 = 0$ com valor objetivo 771.

(*Branching*) Como a solução da relaxação $R(P_2)$ não é factível para o problema original, um limite inferior não foi obtido e temos que quebrar o problema P_2 em dois subproblemas. Assim, obtemos $P_{21} = P_2 \cap \{x : x_3 \leq 2\}$ e $P_{22} = P_2 \cap \{x : x_3 \geq 3\}$, resultando na árvore *branch-and-bound* dada na Figura A.6.

(*Bounding*) Resolvendo a relaxação $R(P_{22})$ detectamos que o problema é infactível, portanto o nó correspondente ao problema P_{22} é fechado devido a infactibilidade.

(*Bounding*) Resolvendo a relaxação $R(P_1)$ obtemos o limite superior 1052 e a solução $x_1 = 1$, $x_2 = 0$, $x_3 = 4.4089$, $y_1 = 5.4838$ e $y_2 = 1.4851$ que viola a restrição de integralidade das variáveis x.

(*Branching*) Quebramos P_1 em dois subproblemas $P_{11} = P_1 \cap \{x : x_3 \leq 4\}$ e $P_{12} = P_1 \cap \{x : x_3 \geq 5\}$. A árvore *branch-and-bound* resultante aparece na Figura A.7. Note que o limite superior do nó P passa de 1165 para $1052 = \max\{1052, 771\}$.

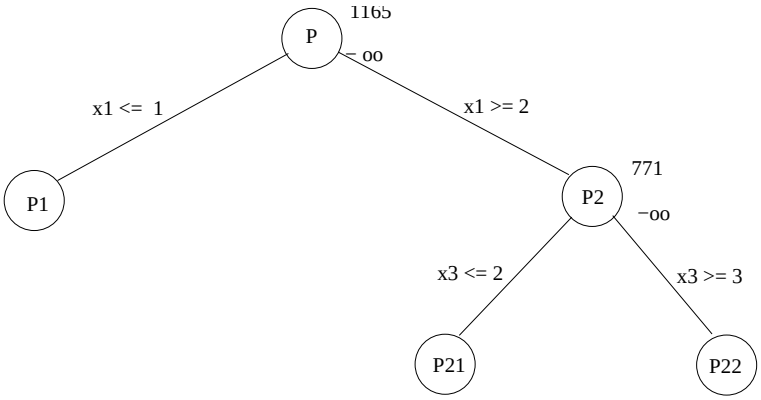


Figura A.6: Árvore *branch-and-bound* parcial

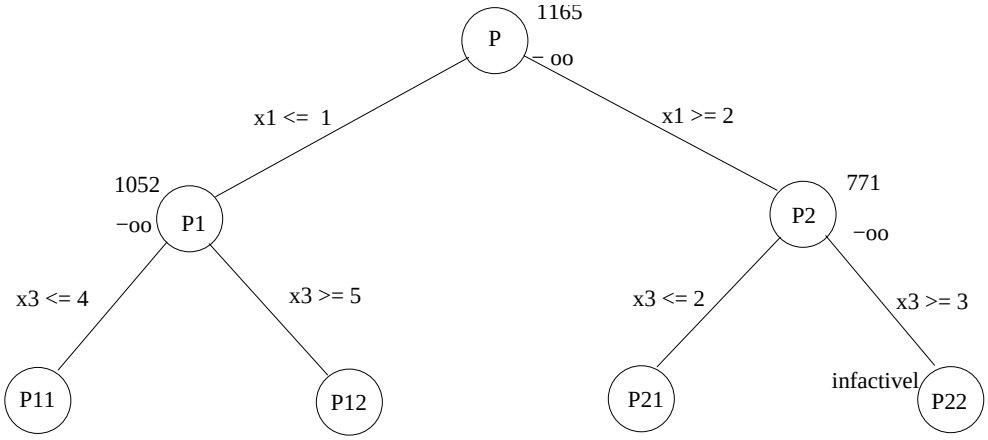


Figura A.7: Árvore *branch-and-bound* parcial

(*Bounding*) Resolvendo a relaxação $R(P_{12})$ verificamos que o problema P_{12} é infactível.

(*Bounding*) Resolvendo a relaxação $R(P_{21})$ produzimos o limite superior 659 para a subárvore com raiz em P_{21} e a solução $x_1 = 2$, $x_2 = 0.1479$, $x_3 = 2$, $y_1 = 3.2248$ e $y_2 = 0$. Uma vez que a solução obtida viola a restrição de integralidade, teremos de gerar subproblemas. Podemos também reduzir o limite superior do nó P_2 de 771 para 659.

(*Branching*) Quebrando o problema P_{21} obtemos os subproblemas $P_{211} = P_{21} \cap \{x : x_2 \leq 0\}$ e $P_{212} = P_{21} \cap \{x : x_2 \geq 1\}$. A árvore *branch-and-bound* resultante é apresentada na Figura A.8.

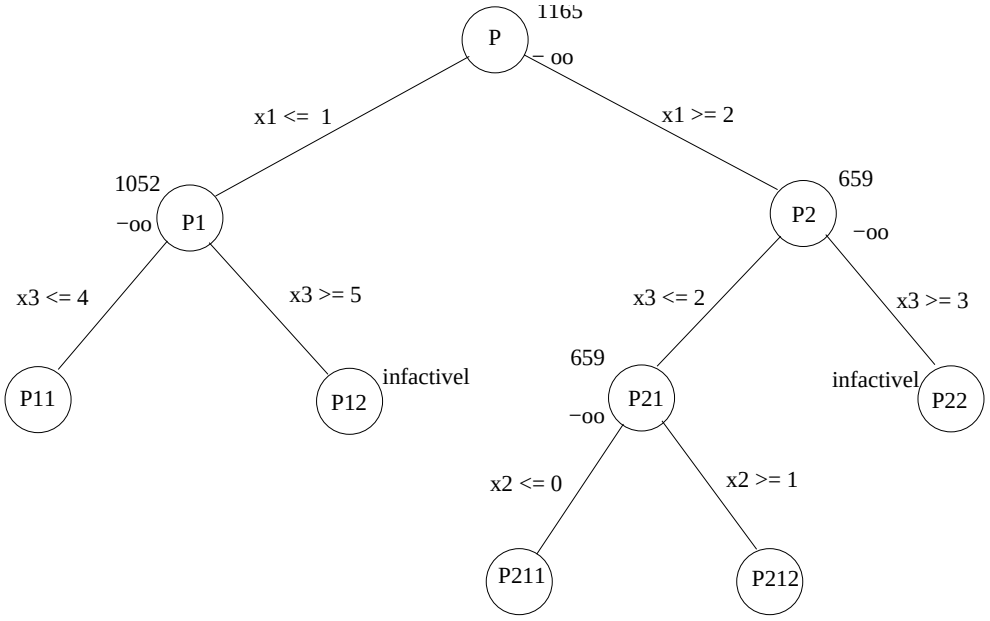


Figura A.8: Árvore *branch-and-bound* parcial

(*Bounding*) Resolvendo a relaxação $R(P_{212})$ obtemos um problema infactível, que nos leva a cortar o nó P_{212} por infactibilidade.

(*Bounding*) Resolvendo a relaxação $R(P_{211})$ obtemos a solução $x_1 = 2.145$, $x_2 = 0$, $x_3 = 2$, $y_1 = 3.2009$ e $y_2 = 0$ com valor objetivo 657. Geramos assim um novo limite superior para os nós P_{21} e P_2. Uma vez que a solução obtida é fracionária, novos subproblemas serão gerados.

(*Branching*) Quebrando o problema P_{211} em dois subproblemas, obtemos $P_{2111} = P_{211} \cap \{x : x_1 \leq 2\}$ e $P_{2112} = P_{211} \cap \{x : x \geq 3\}$. Isto resulta na árvore *branch-and-bound* da Figura A.9.

(*Bounding*) Resolvendo a relaxação $R(P_{11})$ obtemos a solução $x_1 = 1$, $x_2 = 0.2768$, $x_3 = 4$, $y_1 = 5.1513$ e $y_2 = 1.0507$ com valor objetivo 991. Logo, o limite superior do nó P_1 é reduzido para 991 e de P para $\max\{991, 657\} = 991$.

(*Branching*) Executando a quebra de P_{11} obtemos $P_{111} = P_{11} \cap \{x : x_2 \leq 0\}$ e $P_{112} = P_{11} \cap \{x : x_2 \geq 1\}$. A Figura ilustra a árvore *branch-and-bound* resultante

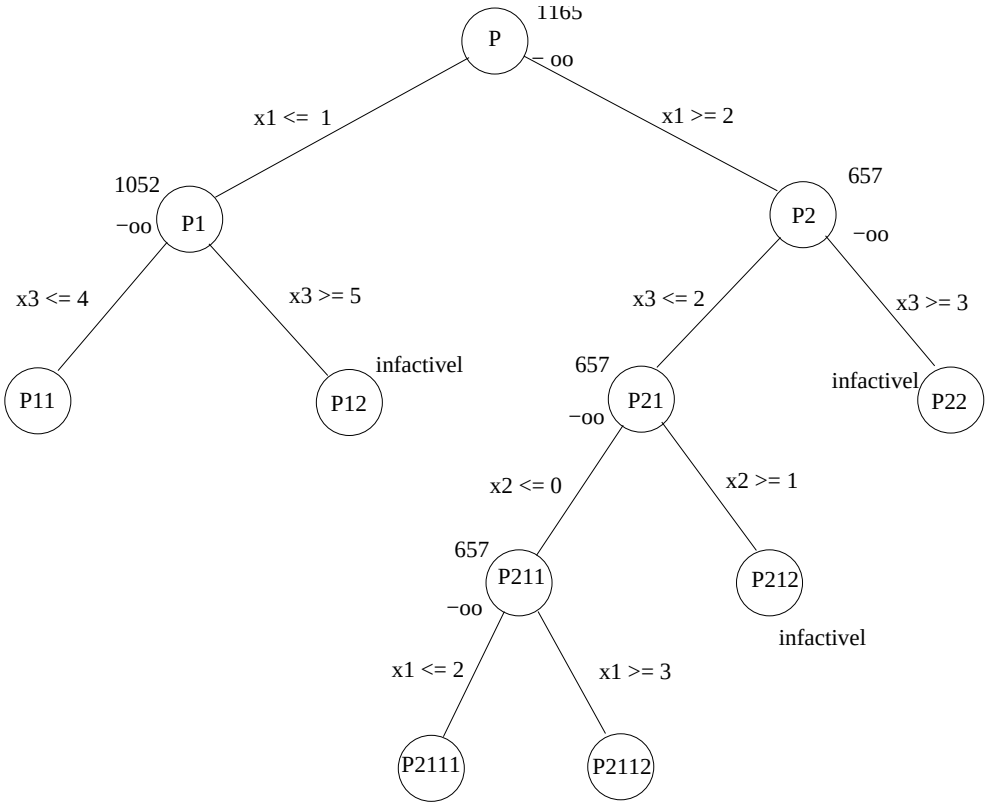


Figura A.9: Árvore *branch-and-bound* parcial

A.10.

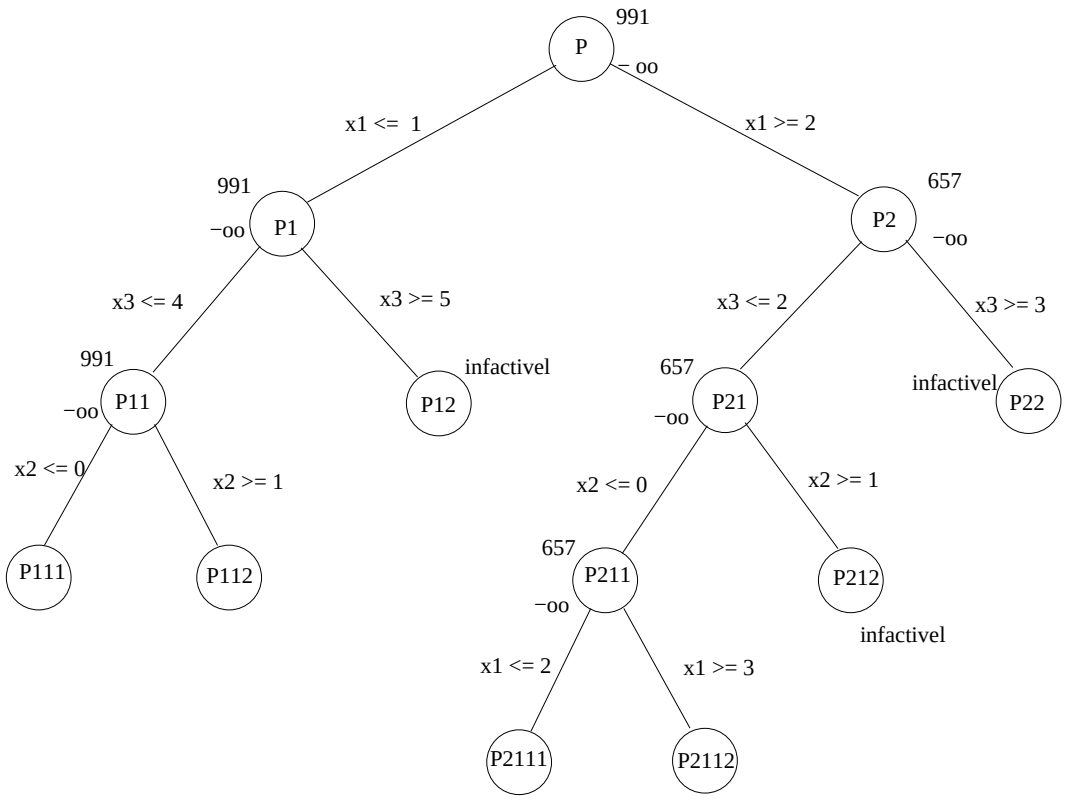


Figura A.10: Árvore *branch-and-bound* parcial

(*Bounding*) Resolvendo a relaxação $R(P_{111})$ obtemos uma solução inteira $x_1 = 1$, $x_2 = 0$, $x_3 = 4$, $y_1 = 5.0702$ e $y_2 = 1.693$ com valor objetivo 982. Portanto os nós P_{2111} e P_{2112} são cortados por limite superior (657) inferior ao limite inferior (982), restando apenas o nó P_{112}. A árvore *branch-and-bound* corrente é dada na Figura A.11.

(*Bounding*) O único subproblema restante é P_{112}. Resolvendo a relaxação $R(P_{112})$ obtemos uma solução fracionária com valor objetivo 824 que, por sua vez, é cortada por limite. Já encontramos um limite inferior com valor superior a 824.

(*Finalização*) A solução ótima é portanto a solução encontrada ao resolvermos o nó P_{111}: $x_1 = 1$, $x_2 = 0$, $x_3 = 4$, $y_1 = 5.0702$ e $y_2 = 1.693$ com valor objetivo 981.6 (tinha sido arredondado para 982).

EX 11.3: Abaixo segue os resultados das tarefas.

i. Ilustramos apenas a árvore *branch-and-bound* final.

As seguintes soluções foram produzidos nos nós da árvore: $(x_1^0, x_2^0) = (5/3, 10.3)$, $(x_1^1, x_2^1) = (1, 4)$, $(x_1^2, x_2^2) = (2, 2.85)$, $(x_1^3, x_2^3) = (2.6, 2)$, $(x_1^5, x_2^5) = (2, 2)$, $(x_1^6, x_2^6) = (3, 1.42837)$, $(x_1^7, x_2^7) = (3.3, 1)$,$(x_1^9, x_2^9) = (3, 1)$, $(x_1^{10}, x_2^{10}) = (4, 0)$.

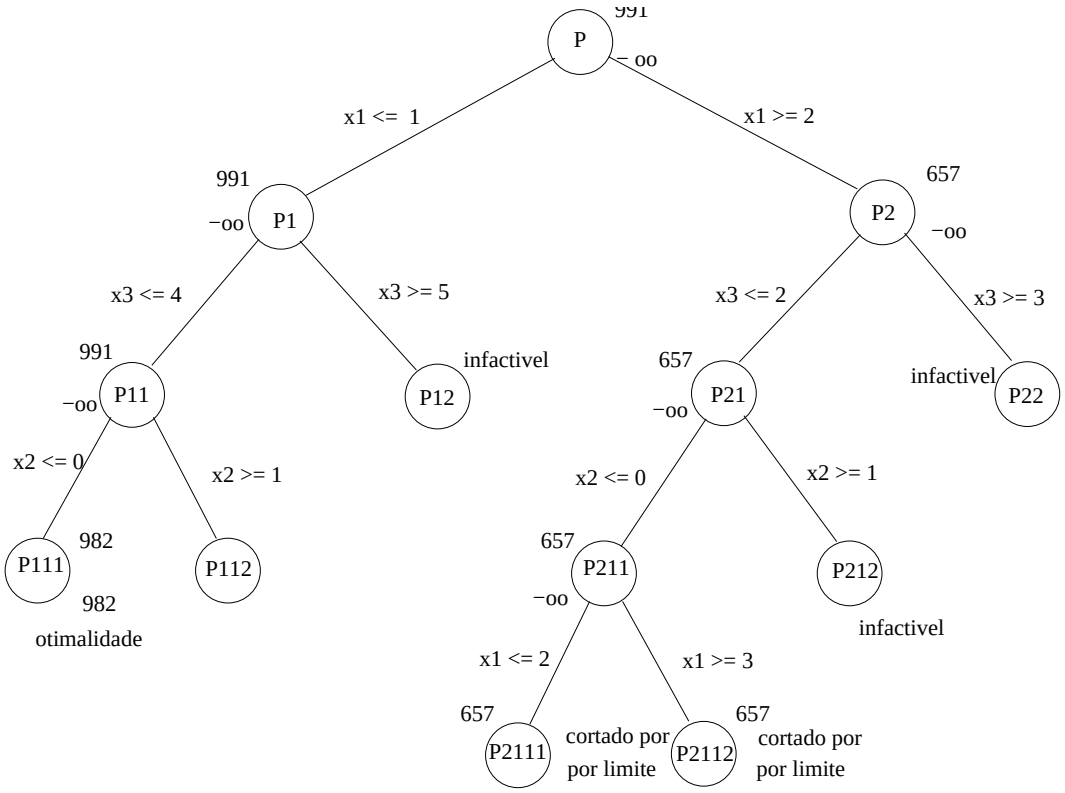


Figura A.11: Árvore *branch-and-bound* parcial

ii. O subproblema Lagrangeano é dado por:

$$\begin{aligned}\mathcal{L}(u_1,u_2) = \text{Maximize} \quad & 17x_1 + 12x_2 \\ & +u_1(40 - 10x_1 - 7x_2) + u_2(5 - x_1 - x_2) \\ \text{Sujeito a :} \quad & \\ & x_1 \leq 4 \\ & x_2 \leq 5 \\ & x_1, x_2 \geq 0 \\ & x_1, x_2 \in \mathbb{Z}\end{aligned}$$

ou, equivalentemente,

$$\begin{aligned}\mathcal{L}(u_1,u_2) = \text{Maximize} \quad & (17 - 10u_1 - u_2)x_1 + (12 - 7u_1 - u_2)x_2 \\ & +40u_1 + 5u_2 \\ \text{Sujeito a :} \quad & \\ & x_1 \leq 4 \\ & x_2 \leq 5 \\ & x_1, x_2 \geq 0 \\ & x_1, x_2 \in \mathbb{Z}\end{aligned}$$

O Lagrangeano dual é dado por:

$$\begin{aligned}LD: \quad \text{Minimize} \quad & \mathcal{L}(u_1,u_2) \\ & u_1, u_2 \geq 0\end{aligned}$$

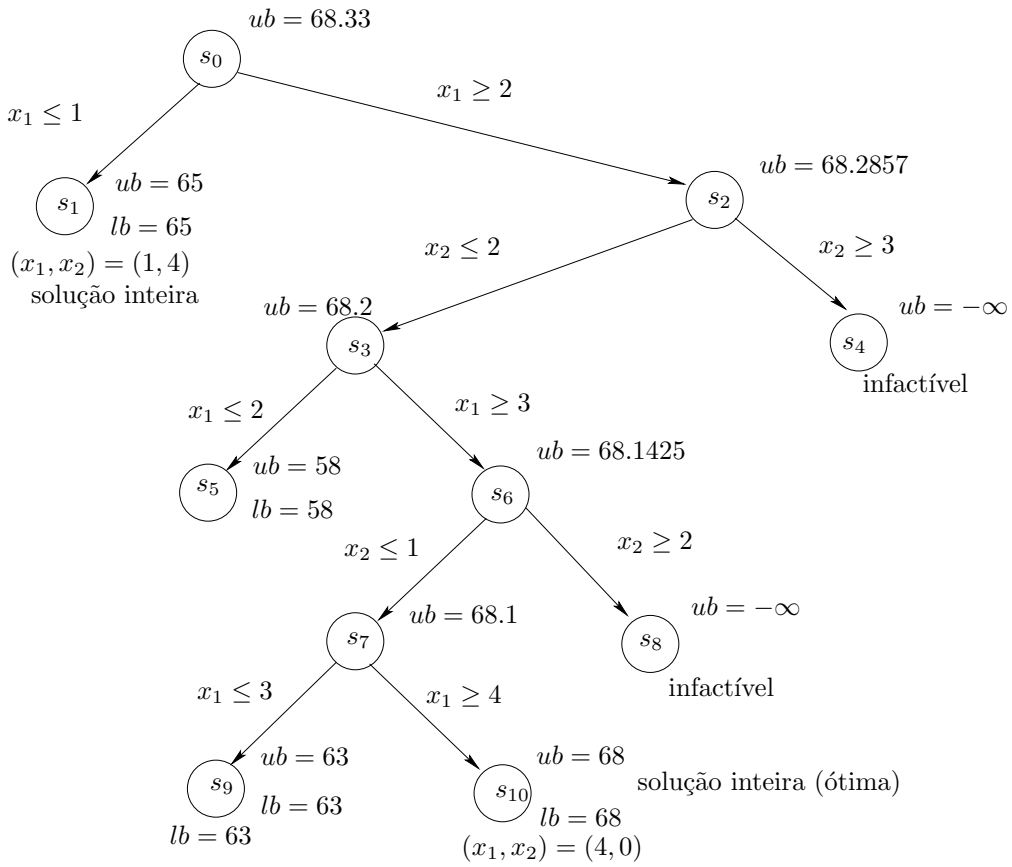


Figura A.12: Árvore *branch-and-bound* parcial

Para $u_1 = 1/2$ e $u_2 = 1$, temos que: a) $(17 - 10u_1 - u_2) = 11 \Rightarrow x_1 = 4$; b) $(12 - 7u_1 - u_2) = 7.5 \Rightarrow x_2 = 5$. Portanto $\mathcal{L}(u_1, u_2) = 11 \times 4 + 7.5 \times 5 + 40/2 + 5 = 106.5$. Observe que $\mathcal{L}(u_1, u_2)$ define um limite superior para o problema inteiro em consideração.

A relaxação contínua corresponde ao problema do nó s_0 da árvore *branch-and-bound*. Logo o limite superior obtido com esta relaxação é 68.33.

A.12 Algoritmos de Planos de Corte e Desigualdades Fortes

EX 12.1: i. $X = \{x \in \mathbb{B}^2 : 3x_1 - 4x_2 \leq 1\}$.

Podemos obter a seguinte desigualdade:

$$\begin{array}{rcll} \left\{ \begin{array}{rcl} 3x_1 - 4x_2 & \leq & 1 \\ x_2 & \leq & 1 \end{array} \right. & & \Rightarrow & 3x_1 - 3x_2 \leq 2 \\ & & \Rightarrow & x_1 - x_2 \leq 2/3 \\ & & \Rightarrow & x_1 - x_2 \leq \lfloor 2/3 \rfloor \\ & & \Rightarrow & x_1 - x_2 \leq 0 \end{array}$$

Basta verificar graficamente que obtemos $conv(X)$ introduzindo a restrição $x_1 \leq x_2$. Note que $X = \{(0,0),(0,1),(1,1)\}$.

ii. $X = \{(x,y) \in \mathbb{R}_+ \times \mathbb{B} : x \leq 20y, x \leq 7\}$.

Adicionando a restrição $x \leq 7y$ pode-se verificar graficamente que se obtém $conv(X)$. Note que $X = \{(0,0)\} \cup \{(x,1) : 0 \leq x \leq 7\}$.

iii. $X = \{(x,y) \in \mathbb{R}_+ \times \mathbb{Z}_+ : x \leq 6y, x \leq 16\}$.

Introduzindo a restrição $x \leq 12 + 4(y-2) = 4y + 4$ à formulação obtemos $conv(X)$. Observe que $X = \{(0,0)\} \cup \{(x,1) : 0 \leq x \leq 6\} \cup \{(x,2) : 0 \leq x \leq 12\} \cup \{(x,3) : 0 \leq x \leq 16\}$.

Ex 12.2: i. $X = \{(x,y) \in \mathbb{R}^2_+ \times \mathbb{B} : x_1 + x_2 \leq 2y, x_1 \leq 1; x_2 \leq 1\}$ e $(x_1, x_2, y) = (1, 0, 1/2)$.

A desigualdade $x_1 \leq y$ é válida e "corta" a solução fracionária. Introduzindo esta restrição à formulação e resolvendo a relaxação linear não obteremos a mesma solução fracionária, com sorte uma solução factível será produzida.

ii. $X = \{(x,y) \in \mathbb{R}_+ \times \mathbb{Z}_+ : x \leq 9, x \leq 4y\}$ e $(x,y) = (9, 9/4)$

A desigualdade $x \leq 8 + (y-2) \Leftrightarrow x \leq 6 + y$ é válida e corta a solução corrente, pois $9 = x \not\leq 6 + y = 6 + 9/4 = 8.25$.

iii. $X = \{(x_1, x_2, y) \in \mathbb{R}^2_+ \times \mathbb{Z}_+ : x_1 + x_2 \leq 25, x_1 + x_2 \leq 8y\}$ e $(x_1, x_2, y) = (20, 5, 25/8)$.

A desigualdade $x_1 + x_2 \leq 25 + (y-4) \Leftrightarrow x_1 + x_2 \leq 21 + y$ é válida para X. Além disso a solução fracionária viola esta restrição pois $25 = 20 + 5 = x_1 + x_2 \not\leq 21 + y = 21 + 25/8 = 24.125$.

iv. $X = \{x \in \mathbb{Z}^5_+ : 9x_1 + 12x_2 + 8x_3 + 17x_4 + 13x_5 \geq 50\}$ e $x = (0, 25/6, 0, 0, 0)$.

Dividindo a desigualdade por 12, produzimos uma desigualdade válida:

$$\begin{array}{rcll} \lceil \frac{9}{12} \rceil x_1 + \lceil \frac{12}{12} \rceil x_2 + \lceil \frac{8}{12} \rceil x_3 + \lceil \frac{17}{12} \rceil x_4 + \lceil \frac{13}{12} \rceil x_5 & \geq & \frac{50}{12} \Rightarrow \\ x_1 + x_2 + x_3 + 2x_4 + 2x_5 & \geq & \lceil \frac{50}{12} \rceil \Rightarrow \\ x_1 + x_2 + x_3 + 2x_4 + 2x_5 & \geq & 5 \end{array}$$

Observe que a última desigualdade é válida por construção e "corta" a solução fracionária corrente, pois $25/6 \not\geq 5$.

v. $X = \{x \in \mathbb{Z}^4_+ : 4x_1 + 8x_2 + 7x_3 + 5x_4 \leq 33\}$ e $x = (0, 0, 33/7, 0)$.
Dividindo a desigualdade por 7 e realizando arredondamento obtemos:

$$\begin{aligned} \frac{4}{7}x_1 + \frac{8}{7}x_2 + \frac{7}{7}x_3 + \frac{5}{7}x_4 \leq \frac{33}{7} \quad &\Rightarrow \quad \lfloor\frac{4}{7}\rfloor x_1 + \lfloor\frac{8}{7}\rfloor x_2 + \lfloor\frac{7}{7}\rfloor x_3 + \lfloor\frac{5}{7}\rfloor x_4 \leq \lfloor\frac{33}{7}\rfloor \\ &\Rightarrow \quad x_2 + x_3 \leq 4 \end{aligned}$$

Observe que a desigualdade acima "corta" a solução fracionária, pois $\frac{33}{7} = 4.7143 = x_2 + x_3 \not\leq 4$.

EX 12.3: Prove que a desigualdade $y_2 + y_3 + 2y_4 \leq 6$ é válida para $X = \{y \in \mathbb{Z}^4_+ : 4y_1 + 5y_2 + 9y_3 + 12y_4 \leq 34\}$.

Dividindo a desigualdade por 5 e realizando arredondamento obtemos:

$$\begin{aligned} \frac{4}{5}y_1 + \frac{5}{5}y_2 + \frac{9}{5}y_3 + \frac{12}{5}y_4 \leq \frac{34}{5} \quad &\Rightarrow \quad \lfloor\frac{4}{5}\rfloor y_1 + \lfloor\frac{5}{5}\rfloor y_2 + \lfloor\frac{9}{5}\rfloor y_3 + \lfloor\frac{12}{5}\rfloor y_4 \leq \lfloor\frac{34}{5}\rfloor \\ &\Rightarrow \quad y_2 + y_3 + 2y_4 \leq 6 \end{aligned}$$

EX 12.4: As iterações do algoritmo de planos de corte seguem abaixo.

Iteração 1: Introduzindo as variáveis de folgas w_1, w_2 e w_3, e resolvendo a relaxação linear obtemos o dicionário:

$$\begin{array}{lrcrcrcrcl} \text{Max} & 68 + \frac{1}{3} & & & - & \frac{35}{21}w_1 & - & \frac{1}{3}w_2 & & \\ \text{S.a :} & & & & & & & & & \\ & x_1 & & & + & \frac{1}{3}w_1 & - & \frac{7}{3}w_2 & & = \frac{5}{3} \\ & & x_2 & - & \frac{1}{3}w_1 & + & \frac{10}{3}w_2 & & & = \frac{10}{3} \\ & & & - & \frac{1}{3}w_1 & + & \frac{7}{3}w_2 & + & w_3 & = \frac{7}{3} \end{array}$$

Podemos então gerar um corte de Gomory na primeira linha:

$$\begin{aligned} x_1 + \lfloor\frac{1}{3}\rfloor w_1 - \lfloor\frac{7}{3}\rfloor w_2 \leq \lfloor\frac{5}{3}\rfloor \quad &\Rightarrow \quad x_1 - 3w_2 \leq 1 \\ &\Rightarrow \quad x_1 - 3w_2 - (x_1 + \frac{1}{3}w_1 - \frac{7}{3}w_2) \leq 1 - \frac{5}{3} \\ &\Rightarrow \quad -\frac{1}{3}w_1 - \frac{2}{3}w_2 \leq \frac{2}{3} \\ &\Rightarrow \quad -\frac{1}{3}w_1 - \frac{2}{3}w_2 + w_4 = -\frac{2}{3} \qquad \text{(A.4)} \end{aligned}$$

Iteração 2: Resolvendo a relaxação linear após introduzir o corte (A.4), obtemos uma solução inteira: $x_1 = 4$, $x_2 = 0$. O valor da função objetivo para esta solução é 68.

A.13 Programação Lógica

A.14 Programação Dinâmica: Domínio Discreto

EX 14.1: Abaixo damos uma implementa em Matlab do algoritmo.

```
%-------------------------------------------------------
% asm.m -- Approximate String Matching
%-------------------------------------------------------

% Input: 1) sp  - pattern string
%        2) st  - text string
%        3) tym - type of matching.
%           tym=1 for perfect matching
%           tym=2 for matching anywhere in the text
% Output: 1) m x n matrix D, where m = length(sp) and
%            n = length(st).
%            Entry D[i,j] is the minimum number of differences
%            between sp(1:i) and st(1:j)
%         2) the matching cost (number of deletions, substitutions,
%            and insertions)

function [D, d] = asm(sp,st,tym)

m = length(sp); n = length(st); D = [];
for j=0:n
   if tym==1
      D(0+1,j+1) = j;
   else
      D(0+1,j+1) = 0;
   end
end
for i=0:m
    D(i+1,0+1) = i;
end

for i=1:m
   for j=1:n
      if sp(i)==st(j)
         d1 = D(i-1+1,j-1+1);
      else
         d1 = D(i-1+1,j-1+1) + 1;
      end
      d2 = D(i-1+1,j+1)+1;
      d3 = D(i+1,j-1+1)+1;
      D(i+1,j+1) = min([d1, d2, d3]);
   end
end

if tym==1
   d = D(m+1,n+1);
end
if tym==2
   d = D(m+1,0+1);
   for j=1:n
      if D(m+1,j+1)<d
         d = D(m+1,j+1);
```

```
        end
    end
end
```

Para o padrão $P = abcdefghijkl$ e texto $T = bcdefffghixkl$, podemos executar os comandos Matlab abaixo para emparelhamento perfeito e flutuante, respectivamente:

```
sp = 'abcdefghijkl'
st = 'bcdeffghixkl'
[D, d] = asm(sp,st,1); % perfect matching
[D, d] = asm(sp,st,1); % matching anywhere in the text
```

Para o caso de emparelhamento perfeito a tabela D é indêntica aos resultados apresentados na Figura 14.2.

Se modificarmos o padrão para $P = fg$ e mantivermos o mesmo texto, então obteremos $D[i, j]$ conforme Tabela A.1 para o caso de emparelhamento perfeito, e $D[i, j]$ conforme Tabela A.2 para o caso de emparelhamento flutuante. Note que no primeiro caso o custo de edição é 10, enquanto que no segundo caso o custo é nulo pois o padrão $P = fg$ aparece no texto T.

Tabela A.1: Tabela $D[i, j]$ para emparelhamento perfeito

	T	b	c	d	e	f	f	g	h	i	x	k	l
P	0	1	2	3	4	5	6	7	8	9	10	11	12
f	1	1	2	3	4	4	5	6	7	8	9	10	11
g	2	2	2	3	4	5	5	5	6	7	8	9	10

Tabela A.2: Tabela $D[i, j]$ para emparelhamento flutuante

	T	b	c	d	e	f	f	g	h	i	x	k	l
P	0	0	0	0	0	0	0	0	0	0	0	0	0
f	1	1	1	1	1	0	0	1	1	1	1	1	1
g	2	2	2	2	2	1	1	0	1	2	2	2	2

EX 14.3: Seja $J_{i,N}$ o custo de formatação ótima para a subcadeia $(w_i, \ldots, w_N)$. Podemos desenvolver as recorrências estágio por estágio.

a) $i = N$**:** Temos o valor terminal:

$$J_{N,N} = \begin{cases} 0 & \text{se } b' = A - L_N \geq b \\ |b' - b| & \text{se } b' < b \end{cases}$$

b) $i = N - 1$**:** Temos a recorrência:

$$J_{N-1,N-1} = |b' - b| \text{ onde } b' = A - L_{N-1}$$

$$J_{N-1,N-1} = \min \begin{cases} 0 & \text{se } L_{N-1} + L_N \geq A \text{ and} \\ & \quad b' = (A - L_{N-1} - L_N)/2 \geq b \\ 2|b' - b| & \text{se } L_{N-1} + L_N \leq A \text{ and } b' < b \\ J_{N-1,N} + J_{N,N} & \end{cases}$$

c) $i = N - 2$**:** Temos a recorrência:

$$J_{N-2,N-2} = |b' - b| \text{ onde } b' = A - L_{N-2}$$

$$J_{N-2,N-1} = \min \begin{cases} 2|b' - b| & \text{se } L_{N-2} + L_{N-1} \leq A \text{ and} \\ & \quad b' = (A - L_{N-2} - L_{N-1})/2 \\ J_{N-2,N-2} + J_{N-1,N-1} & \end{cases}$$

$$J_{N-2,N} = \min \begin{cases} 0 & \text{se } L_{N-2} + L_{N-1} + L_N \leq A \text{ and} \\ & \quad b' = (A - L_{N-2} - L_{N-1} - L_N)/3 \geq b \\ 3|b' - b| & \text{se } L_{N-2} + L_{N-1} + L_N \leq A \\ J_{N-2,N-2} + J_{N-1,N} & \\ J_{N-2,N-1} + J_{N,N} & \end{cases}$$

E assim sucessivamente até obtermos $J_{1,N}$. Uma análise inicial nos leva a deduzir que o cômputo de $J_{i,j}$ consome $\Theta(j - i + 1)$ operações elementares. Portanto, o custo total para resolução das recorrências é $\Theta(N^2)$, consumindo também $\Theta(N^2)$ unidades de mémoria para armazenamento das tabelas $J_{i,j}$, $i, j \in \{1, \ldots, N\}$. Podemos melhorar a análise observando que não é necessário computar $J_{i,j}$ se $\sum_{k=i}^{j} L_k > A$, já que neste caso a subcadeia $(w_i, w_{i+1}, \ldots, w_j)$ não pode ser acomodada na linha. Assim, um algoritmo mais eficiente pode ser obtido com base nesta observação, o qual terá complexidade $\Theta(N \min\{A, N\})$, que será mais eficiente quando o número de palavras for maior do que o comprimento da linha. O algoritmo será linear no tamanho da entrada.

EX 14.4: As respostas estão enumeradas abaixo:

a) Seja $g_k(b)$ o comprimento do caminho mais curto de s até k com custo máximo b. Então podemos formular o problema de forma recursiva:

$$\begin{cases} g_s(b) = 0, \text{ para } b = 0, \ldots, B \\ g_k(0) = +\infty, \text{ para todo } k \in V - \{s\} \\ g_k(b) = \min\{g_k(b-1), \min\{g_j(b - c_{jk}) + w_{jk} : (j,k) \in A, c_{jk} \leq b\}\}, \\ \qquad k \in V - \{s\}, b \in \{1, \ldots, B\} \end{cases}$$

b) As recorrências podem ser resolvidas através do algoritmo a seguir. O algoritmo também matém os parâmetros $\pi_k(b)$, o qual nos dá o predecessor de k no caminho mais curto de s até k com custo máximo b.

Algoritmo

Inicialização

Para $b = 0, \ldots, B$ faça
  $g_s(b) \leftarrow 0$
  $\pi_s(b) \leftarrow nil$
Para todo $k \in V - \{s\}$ faça
  $g_k(0) \leftarrow \infty$
  $\pi_k(0) \leftarrow nil$

Recorrência

Para $b = 1, \ldots, B$ faça
  Para todo $k \in V - \{s\}$ faça
    $g_k(b) \leftarrow g_k(b-1)$
    $\pi_k(b) \leftarrow \pi_k(b-1)$
    Para todo $(j,k) \in A$ faça
      Se $c_{jk} \leq b$ e $g_j(b - c_{jk}) + w_{jk} < g_k(b)$
        $g_k(b) \leftarrow g_j(b - c_{jk}) + w_{jk}$
        $\pi_k(b) \leftarrow j$

Para cada valor $b \in \{1, \ldots, B\}$, o algoritmo varre todos os arcos do grafo. Assumindo que existe caminho com origem em s para todo k, a complexidade do algoritmo é $\Theta(B|E|)$. A memória ocupada pelo algoritmo é também da ordem $\Theta(2B|E|)$ necessária para armazenar as tabelas $g_k(b)$ e $\pi_k(b)$.

EX 14.5: Seja $D(i,j)$ o comprimento da maior subsequência em comum para as subcadeias $x(i) = (x_1, \ldots, x_i)$ e $y(j) = (y_1, \ldots, y_j)$. Se $i = 0$ então definimos $x(i) =$ como uma cadeia vazia, o mesmo ocorrendo quando $j = 0$ que implica $y(j) =$. Podemos então colocar o problema de uma forma recursiva:

$$\begin{cases} D(0,j) = 0, j = 0, \ldots, m \\ D(i,0) = 0, i = 0, \ldots, n \\ D(i,j) = \max\{D(i,j-1), D(i-1,j), \\ \qquad\qquad D(i-1,j-1) + 1 \text{ se } x_i = y_j\}, \ i,j > 0 \end{cases} \tag{A.5}$$

A recorrência dada por (A.5) nos leva imediatamente a um algoritmo de programação dinâmica.

Algoritmo

Inicialização

Para $j = 0, \ldots, m$ faça
  $D(0,j) \leftarrow 0$
Para $i = 0, \ldots, n$ faça
  $D(i,0) \leftarrow 0$

Recorrência

Para $j = 1, \ldots, m$ faça
  Para $i = 1, \ldots, n$ faça
    $D(i,j) \leftarrow \min\{D(i-1,j), D(i,j-1)\}$
    Se $x_i = y_j$ então
      $D(i,j) \leftarrow \min\{D(i,j), D(i-1,j-1)+1\}$

A execução do algortimo para as cadeias $x = (A,B,C,B,D,A,B)$ e $y = (B,D,C,A,B,A)$ pode ser observeda na Figura A.13.

A partir do algoritmo de programação dinâmica é imediata a conclusão que a memória utilizada é $\Theta(nm)$ e o tempo de execução, que corresponde ao preenchimento das entradas da tabela $D(i,j)$, é $\Theta(nm)$.

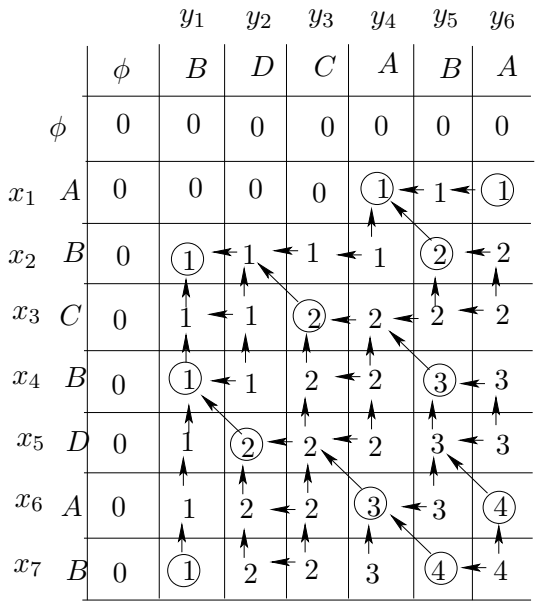


Figura A.13: Exemplo de execução do algoritmo de programação dinâmica para subsequência mais longa de duas cadeias de caracteres.

Para computar as subsequências mais longas em comum entre x e y, podemos construir um grafo direcionado e acíclico, tendo como vértices os pontos de casamento máximo (círculos com valor 4 na Figura A.13), os vértices de casamento perfeito alcançáveis (com círculo) a partir dos vértices de valor 4, e com arcos dados pela relação de "alcançável". Para o exemplo da figura, o grafo correspondente aparece na Figura A.14. Todos os possíveis camimhos de vértices com rótulo 4 até vértices com rótulo 1 definem emparelhamentos que induzem subcadeias mais longas. Note que os arcos são sempre de um vértice com rótulo i para um vértice com rótulo $i-1$. Portanto, todos os caminhos têm o mesmo comprimento, ou seja, o comprimento da subcadeia mais longa.

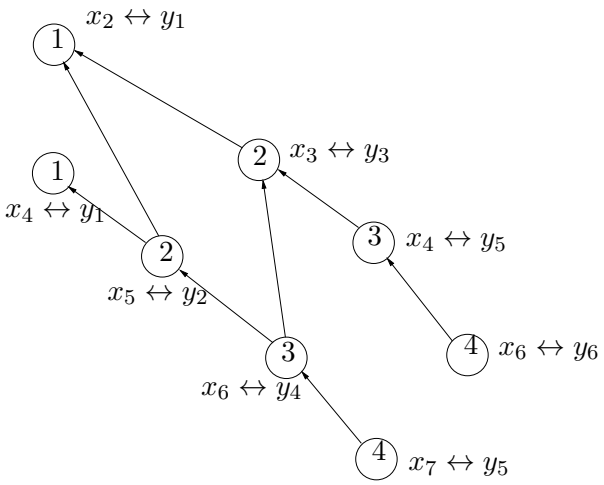


Figura A.14: Grafo para cômputo de todos os possíveis emparelhamentos de subcadeias mais longas.

EX 14.6: Seja $G = (V, E)$ a árvore com a hierarquia da empresa XYZ. Inicialmente, considere o subproblema restrito à subárvore dos descendentes do vértice n:

$$\begin{array}{rcll} J_n^y & = & \text{Maximize} & \sum_{j \in S_n} r_j x_j \\ & & \text{Sujeito a :} & \\ & & & x_i + x_j \leq 1, \ \forall (i,j) \in E[S_n] \\ & & & x_n = y \\ & & & x_i \in \{0,1\}, \ \forall i \in S_n \end{array}$$

onde $S_n = \{j : \text{existe caminho de } n \text{ para } j \text{ em } G\}$ é o conjunto dos descendentes de n, $y = 1$ se o empregado n é convidado e $y = 0$ caso contrário. Note que $n \in S_n$. Defina também:

$$J_n = \max\{J_n^0, J_n^1\}$$

afinal de contas, temos que decidir se convidamos ou não o empregado n. Então o problema da "festa otimizada" pode ser resolvido recursivamente, conforme as recorrências abaixo:

$$\left\{ \begin{array}{rcl} J_n^0 & = & \sum_{j \in A_n} J_j \\ J_n^1 & = & r_n + \sum_{j \in A_n} J_j^0 \\ J_n & = & \max\{J_n^0, J_n^1\} \end{array} \right. \tag{A.6}$$

onde $A_n = \{j : (n, j) \in E\}$ é o conjunto dos descendentes imediatos de n.

Podemos resolver eficientemente as recorrências (A.6) seguindo um ordem topológica reversa de G. Obtemos um algoritmo de programação dinâmica.

Algoritmo

Inicialização

Obtenha uma ordem topológica reversa $I = (i_1, \ldots, i_N)$
dos vértices de G

Para $n = 1, \ldots, N$ faça

$J_n \leftarrow J_n^0 \leftarrow J_n^1 \leftarrow 0$

Recorrência

Para $k = 1, \ldots, N$ faça

$n \leftarrow i_k$

$J_n^0 = \sum_{j \in A_n} J_j$

$J_n^1 = r_n + \sum_{j \in A_n} J_j^0$

$J_n = \max\{J_n^0, J_n^1\}$

Tabela A.3: Resolução das recorrências do algoritmo de programação dinâmica

n	36	35	34	33	32	31	30	29	28	27
J_n^0	0	0	0	0	0	0	0.5932	0	0.6013	0
J_n^1	0.2314	0.3618	0.6013	0.9786	0.8832	0.1613	0.1397	0.6041	0.8976	0.2457
J_n	0.2314	0.3618	0.6013	0.9786	0.8832	0.1613	0.5932	0.6041	0.8976	0.2457
n	26	25	24	23	22	21	20	19	18	17
J_n^0	0	0	0.9786	0	1.0445	0	0	0.5932	1.5017	0
J_n^1	0.8194	0.8571	0.8624	0.3838	0.1118	0.7379	0.1851	1.1659	0.9255	0.9016
J_n	0.8194	0.8571	0.9786	0.3838	1.0445	0.7379	0.1851	1.1659	1.5017	0.9016
n	16	15	14	13	12	11	10	9	8	7
J_n^0	1.0651	0	0.8571	0.9786	0	1.4283	0.7379	2.8527	0.9016	2.0151
J_n^1	0.2405	0.0929	0.2781	1.3284	0.9056	2.0386	0.2468	2.4287	0.0096	2.2616
J_n	1.0651	0.0929	0.8571	1.3284	0.9056	2.0386	0.7379	2.8527	0.9016	2.2616
n	6	5	4	3	2	1				
J_n^0	2.2340	2.7765	3.7543	2.2616	5.0105	11.9527				
J_n^1	1.4419	1.9883	4.5095	2.1511	5.1816	11.9979				
J_n	2.2340	2.7765	4.5095	2.2616	5.1816	11.9979				

Note que uma ordem topológica pode ser obtida em tempo $O(N + M) = O(N)$, onde $N = |V|$ e $M = |E|$. O laço de inicialização é executado em tempo $O(N)$. O laço de recorrência percorre cada vértice precisamente uma vez. Para cada vértice n, são executadas $O(\delta(n) + 1)$ operações sendo $\delta(n) = |A_n|$. Portanto, o custo computacional do laço é da ordem de $O(\sum_{n \in V} \delta(n)) + O(N) = O(N)$. Concluímos que o algoritmo executa em tempo linear $\Theta(N)$.

A quantidade de memória é precisamente o número de entradas necessárias para armazenar as tabelas, sendo esta da ordem de $\Theta(3N) = \Theta(N)$.

Para garantir que o presidente será convidado, basta utilizar a solução cujo valor é J_1^1, segundo a qual o presidente (vértice 1) é convidado.

Na solução ótima para o problema o presidente é convidado, pois $J_1^1 > J_1^0$ conforme a Tabela A.3. Não é difícil construir um algoritmo que percorre esta tabela e produz a lista de convidados. Os seguintes convidados constam da lista: 1, 7, 11, 12, 13, 17, 19, 20, 21, 25, 26, 27, 28, 29, 31, 32, 33, 35 e 36.

EX 14.7: Daremos uma resposta informal ao problema. Seja $m \in \{1, \ldots, 21\}$ o número de palitos que restam sobre a mesa. Eliminamos o contador de iteração k pois a política de controle ótima depende apenas do número de palitos restantes, sendo independente de quantas iterações foram realizadas. Seja $\pi(m) \in \{1, \ldots, 3\}$ a decisão ótima quando há m palitos sobre a mesa. Seja também $J(m) \in \{0, 1\}$ o ganho recebido pelo jogador, onde $J(m) = 1$ se o jogador ganhará com certeza, e $J(m) = 0$ caso contrário. Podemos calcular a política de controle ótima $\pi(m)$ resolvendo para $m = 1, 2, \ldots, 21$, conforme indicam as tabelas A.4 e A.5. O asterisco indica que qualquer valor válido, $\{1, 2, 3\}$, pode ser selecionado sem afetar o resultado final.

Tabela A.4: Jogos dos palitos

m	1	2	3	4	5	6	7	8	9	10	11	12	13
$\pi(m)$	1	1	2	3	*	1	2	3	*	1	2	3	*
$J(m)$	0	1	1	1	0	1	1	1	0	1	1	1	0

Tabela A.5: Jogos dos palitos

m	14	15	16	17	18	19	20	21
$\pi(m)$	1	2	3	*	1	2	3	*
$J(m)$	1	1	1	0	1	1	1	0

A.15 Programação Dinâmica: Domínio Contínuo

EX 15.1: O problema pode ser colocado de forma recursiva em três estágios:

$$\begin{array}{rcl} J_2(x_2) & = & r(x_2 - T)^2 \\ J_1(x_1) & = & \underset{u_1}{\text{Min}}\ u_1^2 + J_2(x_2) \\ & & \text{S.a: } x_2 = (1-\alpha)x_1 + \alpha u_1 \\ J_0(x_0) & = & \underset{u_0}{\text{Min}}\ u_0^2 + J_1(x_1) \\ & & \text{S.a: } x_1 = (1-\alpha)x_0 + \alpha u_0 \end{array}$$

No estágio terminal não há o que otimizar, logo podemos encontrar a política de controle ótima $\mu_1^*(x_1)$ dependente do estado no penúltimo estágio.

$$\text{Min } f_1(x_1) = u_1^2 + r[(1-\alpha)x_1 + \alpha u_1 - T]^2$$

Uma vez que f_1 é uma função convexa em y_1, podemos encontrar a solução ótima quando a derivada é nula.

$$\begin{array}{rcl} \dfrac{d}{u_1} f_1 & = & \dfrac{d}{u_1}\left\{u_1^2 + r[(1-\alpha)x_1 + \alpha u_1 - T]^2\right\} \\ & = & u_1 + r\alpha(1-\alpha)x_1 + r\alpha^2 u_1 - r\alpha T \end{array}$$

Fazendo $df_1/du_1 = 0$, concluímos que:

$$\begin{array}{rcl} \mu_1^*(x_1) & = & \dfrac{r\alpha[T - (1-\alpha)x_1]}{1 + ra^2} \\ J_1^*(x_1) & = & u_1^2 + r(x_2 - T)^2 \\ & = & \mu_1^*(x_1)^2 + r\left[(1-\alpha)x_1 + \alpha\mu_1^*(x_1) - T\right]^2 \\ & = & \dfrac{r[(1-\alpha)x_1 - T]^2}{1 + r\alpha^2} \end{array}$$

Procedendo ao primeiro estágio, podemos obter $J_0(x_0)$ com base na política ótima a ser aplicada no estágio seguinte:

$$\begin{array}{rcl} J_0(x_0) & = & u_0^2 + J_1(x_1) \\ & = & u_0^2 + \dfrac{r[(1-\alpha)x_1 - T]^2}{1 + r\alpha^2} \end{array}$$

Uma vez que J_0 é convexa, podemos calcular o valor mínimo fazendo a derivada em relação a u_0 assumir o valor nulo. A derivada de J_0 é:

$$\frac{d}{du_0} J_0 = 2u_0 + \frac{2r}{1 + ra^2}[(1-\alpha)^2 x_0 + \alpha(1-\alpha)u_0 - T]\alpha(1-\alpha)$$

e fazendo dJ_0/du_0 nula, calculamos a lei de controle ótima para o primeiro estágio:

$$\mu_0^*(x_0) = \frac{r\alpha(1-\alpha)}{1 + r\alpha^2(1 + (1-\alpha)^2)}[T - (1-a)^2 x_0]$$

Consequentemente, obtivemos uma política de controle ótima $\pi^* = (\mu_0^*, \mu_1^*)$.

EX 15.3: Considerações inciais:

- Seja $x_k \in \{t, b\}$ o estado da máquina no início da semana k, on t indica que a máquina está trabalhando e q, máquina quebrada.
- Seja $u_k(q) \in \{r, s\}$ ação no ínicio da semana k se o estado da máquina é *quebrada*, onde r indica reparo e s substituição por uma máquina nova.
- Seja $u_k(t) \in \{m, n\}$ a ação no início da semana k se o estado da máquina é *trabalhando*), onde m indica manutenção preventiva e n indica que nenhuma medida será tomada.
- $J_k(x_k)$ o lucro máximo a partir da semana k se a máquina inicia a semana no estado x_k.

Vamos aplicar o algoritmo de programação dinâmica a partir da última semana e progredir até a primeira semana.

Semana $k = 4$: Consideramos os dois possíveis estados:

a) $x_k = t$**:** O custo ótimo é dado por:

$$\begin{array}{rcl} J_4(t) & = & \max\{ \underbrace{0.6 \times 80 + 0.4 \times -20}_{\text{manutenção}}, \underbrace{0.3 \times 100 + 0.7 \times 0}_{\text{sem prevenção}} \} \\ & = & \max\{ \quad 40, \quad\quad 30 \quad\quad \} = 40 \end{array}$$

Portanto, $J_4(t) = 40$ e $\mu_4(t) = m$.

b) $x_k = q$**:** O custo ótimo é dado por:

$$\begin{array}{rcl} J_4(q) & = & \max\{ \underbrace{0.6 \times 60 + 0.4 \times -40}_{\text{reparo}}, \underbrace{\quad -50 \quad}_{\text{substituição}} \} \\ & = & \max\{ \quad 20, \quad\quad -50 \quad\quad \} = 20 \end{array}$$

Portanto, $J_4(q) = 20$ e $\mu_4(q) = r$.

Semana $k = 3$: Consideramos os dois possíveis estados:

a) $x_k = t$**:** O custo ótimo é dado por:

$$\begin{array}{rcll} J_3(t) & = & \max\{ & 0.6 \times (80 + J_4(t)) + 0.4 \times (-20 + J_4(q)), \quad \text{manut} \\ & & & 0.3 \times (100 + J_4(t)) + 0.7 \times (0 + J_4(q)) \quad \} \quad \text{n} \\ & = & \max\{ & 0.6 \times (80 + 40) + 0.4 \times (-20 + 20), \\ & & & 0.3 \times (100 + 40) + 0.7 \times 20 \quad \} \\ & = & \max\{ & 72,56 \quad \} = 72 \end{array}$$

Portanto, $J_3(t) = 72$ e $\mu_3(t) = m$.

b) $x_k = q$**:** O custo ótimo é dado por:

$$\begin{array}{rcll} J_3(q) & = & \max\{ & 0.6 \times (60 + J_4(t)) + 0.4 \times (-40 + J_4(q)), \quad \text{rep} \\ & & & -50 + J_4(t) \quad\quad\quad\quad \} \quad\quad\quad\

Semana $k = 2$: Se iniciamos a semana com a máquina trabalhando, $x_k = t$, então $J_2(t) = 104$ e $\mu_2(t) = m$. Se iniciamos com a máquina quebrada, $x_k = q$, então $J_2(q) = 84$ e $\mu_2(q) = r$.

Semana $k = 1$: Iniciamos a primeira semana com a máquina trabalhando, $x_k = t$, o que nos leva a $J_1(t) = 136$ e $\mu_1(t) = m$.

EX 15.6: Abaixo seguem as soluções item a item.

i. O sistema de equações diferenciais pode ser colocado na forma:

$$\begin{bmatrix} (M+m) & ml \\ 1 & 2l \end{bmatrix} \begin{bmatrix} \ddot{y} \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} u \\ 2g\theta \end{bmatrix} \tag{A.7}$$

Seja A a matriz à esquerda da equação acima. Podemos calcular A^{-1} fazendo $A^{-1} = adj(A)/det(A)$:

$$A^{-1} = \frac{1}{(M+m)2l - ml} \begin{bmatrix} 2l & -ml \\ -1 & (M+m) \end{bmatrix}$$

Pré-multiplicando (A.7) por A^{-1}, obtemos:

$$\begin{bmatrix} \ddot{y} \\ \ddot{\theta} \end{bmatrix} = \frac{1}{(2M+m)l} \begin{bmatrix} 2l & -ml \\ -1 & (M+m) \end{bmatrix} \begin{bmatrix} u \\ 2g\theta \end{bmatrix} \tag{A.8}$$

Fazendo $x_1 = y$, $x_2 = \dot{y}$, $x_3 = \theta$ e $x_4 = \dot{\theta}$, podemos colocar (A.8) na forma:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{-2mg}{2M+m} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & \frac{2g(M+m)}{(2M+m)l} & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{2}{2M+m} \\ 0 \\ -\frac{1}{(2M+m)l} \end{bmatrix} u \tag{A.9}$$

Substituindo as constantes com grandeza físicas em (A.10) obtemos os ssitema de equações:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & -0.4780 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 20.0780 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} 0 \\ 0.0976 \\ 0 \\ -0.0976 \end{bmatrix} u \tag{A.10}$$

EX 15.7: O problema pode ser modelado como um processo Markoviano conforme Figura A.15, onde S é o estado inicial, S_P (S_I) é o estado intermediário com valor par (ímpar) e F_P (F_I) é o estado terminal com valor par (ímpar). Conhecemos o valor dos estados terminais:

$$\begin{aligned} J(F_P) &= -100 \\ J(F_I) &= 100 \end{aligned}$$

Podemos então calcular o valor ótimo dos estados intermediários:

$$\begin{aligned} J(S_P) &= \max\{\text{rola dado } A,\ \text{rola dado } B\} \\ &= \max\{-110 \times \frac{1}{2} + 100 \times \frac{1}{2}, -110 \times 0.3 + 100 \times 0.7\} \\ &= \max\{-5, 37\} \\ &= 37 \\ J(S_I) &= \max\{\text{rola dado } A,\ \text{rola dado } B\} \\ &= \max\{-110 \times 0.5 + 100 \times 0.5, -110 \times 0.7 + 100 \times 0.3\} \\ &= \max\{-5, -47\} \\ &= -5 \end{aligned}$$

Logo a política ótima para os estados S_P e S_I é:

$$\begin{aligned} \mu(S_P) &= B \\ \mu(S_I) &= A \end{aligned}$$

Continuando o processo recursivo, calculamos o valor do estado inicial S:

$$\begin{aligned} J(S) &= \max\{\text{rola dado } A,\ \text{rola dado } B\} \\ &= \max\{J(S_P) \times \frac{1}{2} + J(S_I) \times \frac{1}{2}, J(S_P) \times 0.3 + J(S_I) \times 0.7\} \\ &= \max\{16, 7.6\} \\ &= 16 \end{aligned}$$

Portanto a ação ótima para o estado inicial é $\mu(S) = A$. Obtemos uma política ótima $\pi = (\mu(S), \mu(S_P), \mu(S_I)) = (A, B, A)$.

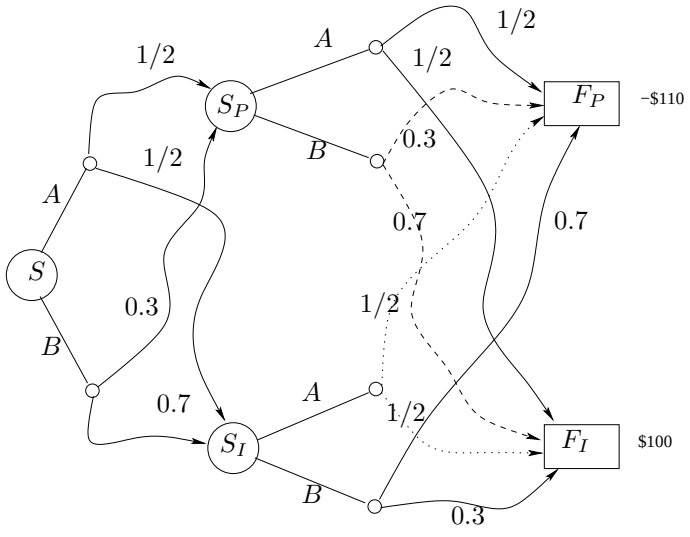


Figura A.15: Processo Markoviando do jogo par-ímpar.

A.16 Programação Não-Linear Sob Restrições: Fundamentos

EX 16.2 Vamos mostrar que o número de soluções que satisfazem as condições KKT de primeira ordem é finito e, portanto, o número de ótimos locais é obrigatoriamente finito. Tomemos o Lagrangeano:

$$\mathcal{L}(x,\lambda) = (x_2 + 100)^2 + 0.01{x_1}^2 - \lambda(x_2 - \cos x_1)$$

Diferenciando $\mathcal{L}$, obtemos:

$$\nabla_{x_1}\mathcal{L} = 0.02x_1 - \lambda \sin x_1 \quad (A.11)$$
$$\nabla_{x_2}\mathcal{L} = 2(x_2 + 100) - \lambda \quad (A.12)$$
$$\nabla_{\lambda}\mathcal{L} = -x_2 + \cos x_1 \quad (A.13)$$

Uma vez que $\nabla_{x_2}\mathcal{L} = 0$, concluímos a partir de (A.12) que $x_2 = \frac{\lambda}{2} - 100$. Substituindo x_2 em (A.11) e (A.13), e de $\nabla\mathcal{L} = 0$, teremos:

$$0.02x_1 - \lambda \sin x_1 = 0 \quad (A.14)$$
$$\cos x_1 - \frac{\lambda}{2} + 100 = 0 \quad (A.15)$$

Evidenciando λ em (A.15), temos $\lambda = 2\cos x_1 + 200$ e depois substituindo em (A.14) deduzimos que:

$$0.02x_1 - 2\cos x_1 \sin x_1 - 200 \sin x_1 = 0 \quad (A.16)$$

A equação (A.16) pode ser colocada na forma:

$$h(x_1) = g(x_1) \quad (A.17)$$

onde $h(x_1) = 0.02x_1$ e $g(x_1) = 2\cos x_1 \sin x_1 + 200 \sin x_1$. Note que $g(x_1)$ é periódica é limitada, sendo $g(x_1) \in [-202, 202]$. Portanto a reta $h(x_1)$ intercepta $g(x_1)$ um número finito de vezes. Daí concluímos que o número de soluções de (A.16) é finito, portanto, o número de soluções KKT também é finito.

EX 16.5: Em princípio podemos utilizar o pacote do Prof. Kunz fazendo $g(x) = f(x)^T f(x) = \|f(x)\|^2$. Note que $g : \mathbb{R}^n \to \mathbb{R}$, sendo g contínua e diferenciável pois f é contínua e diferenciável. Seja $f = [f_1(x) \dots f_n(x)]^T$ com $f_j : \mathbb{R}^n \to \mathbb{R}$. Então $g(x) = 0 \Leftrightarrow f(x)^T f(x) = 0 \Leftrightarrow f_j(x) = 0,\ j = 1, \dots, n \Leftrightarrow x$ é solução de $f(x) = 0$.

EX 16.6: Podemos fazer uso do pacote de software do Prof. Kunz fazendo $h(x) = f'(x)$ e $g(x) = -f''(x)$. Neste caso, se encontrarmos x tal que $h(x) = 0$ e $g(x) < 0$ então x satisfaz as condições suficientes de segunda ordem, $f'(x) = 0$ e $f''(x) > 0$, e deve ser um mínimo local.

EX 16.7: Tomemos inicialmente o Lagrangeano do problema:

$$\mathcal{L}(x,\lambda,\mu) = \sum_{i=1}^{n} c_i x_i + \sum_{i=1}^{n} g_i(x_i) + \lambda(\sum_{i=1}^{n} x_i - b) - \sum_{i=1}^{n} \mu_i x_i \tag{A.18}$$

com $\lambda \geqslant 0$ e $\mu_i \geqslant 0, i = 1, \ldots, n$. Pelas condições KKT, todo x mínimo satisfaz as relações:

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial x_i} = c_i + \frac{\partial g_i(x_i)}{\partial x_i} + \lambda - \mu_i = 0, \quad & i = 1, \ldots, n \\
\sum_{i=1}^{n} x_i - b \leqslant 0 & \\
x_i \geqslant 0, \quad & i = 1, \ldots, n \\
\mu_i \geqslant 0, \quad & i = 1, \ldots, n \\
\lambda \geqslant 0 & \\
\lambda(\sum_{i=1}^{n} x_i - b) = 0 & \\
\mu_i x_i = 0, \quad & i = 1, \ldots, n
\end{aligned}$$

Fazendo

$$\mu_i = c_i + \frac{\partial g_i(x_i)}{\partial x_i} + \lambda$$

e substituindo no sistema KKT, eliminamos μ_i e obtemos:

$$\begin{aligned}
\sum_{i=1}^{n} x_i - b \leqslant 0 & \\
x_i \geqslant 0, \quad & i = 1, \ldots, n \\
c_i + \frac{\partial g_i(x_i)}{\partial x_i} + \lambda \geqslant 0, \quad & i = 1, \ldots, n \\
\lambda \geqslant 0 & \\
\lambda(\sum_{i=1}^{n} x_i - b) = 0 & \\
x_i \left[c_i + \frac{\partial g_i(x_i)}{\partial x_i} + \lambda \right] = 0, \quad & i = 1, \ldots, n
\end{aligned}$$

A.17 Programação Não-Linear Sob Restrições: Fundamentos de Algoritmos

A.18 Programação Quadrática

EX 18.2: Primeiramente, vamos expandir a função o objetivo:

$$f(x) = \frac{1}{2}(x - x_0)^T(x - x_0) = \frac{1}{2}x^T x - x^T x_0 + \frac{1}{2}x_0^T x_0$$

Vamos agora obter o Lagrangeano:

$$\mathcal{L}(x, \lambda) = f(x) - \lambda^T(Ax - b) = \frac{1}{2}x^T x - x^T x_0 + \frac{1}{2}x_0^T x_0 - \lambda^T(Ax - b)$$

sendo $\lambda \in \mathbb{R}^{m\times 1}$. Calculando os gradientes de $\mathcal{L}$ com respeito a x e λ, obtemos:

$$\begin{aligned} \nabla_x \mathcal{L} &= x - x_0 - A^T\lambda \\ \nabla_\lambda \mathcal{L} &= -(Ax - b) \end{aligned}$$

Uma vez que $\nabla\mathcal{L} = 0$, deduzimos que

$$A^T\lambda = x - x_0 \Leftrightarrow AA^T\lambda = A(x - x_0)$$

Uma vez que A tem posto de linha completo, $AA^T \in \mathbb{R}^{m\times m}$ e $rank(AA^T) = m$. Logo,

$$\lambda = (AA^T)^{-1}A(x - x_0) = (AA^T)^{-1}(Ax - Ax_0) = (AA^T)^{-1}(b - Ax_0)$$

como esperado. Portanto, a solução do problema é:

$$\begin{aligned} x &= x_0 + A^T\lambda \\ &= x_0 + A^T(AA^T)^{-1}(b - Ax_0) \end{aligned}$$

EX 18.4: Prof. Kunz está equivocado em sua afirmação pois podem haver equações redundantes, i.e., podem existir linhas de A linearmente dependentes. Não faria sentido resolver o problema se $rank(A) < rank([A|b])$ (i.e, quando não existe solução para $Ax = b$) ou se $rank(A) = rank([A|b]) = n$ (i.e., quando a solução de $Ax = b$ é única). Faz sentido resolver o problema de otimização quando existem infinitas solução: $rank(A) = rank([A|b]) < n$. Considere o problema abaixo:

$$\begin{array}{ll} \text{Minimize} & \|(x_1, x_2)\| \\ \text{Sujeito a :} & \\ & \begin{array}{rcrcr} x_1 & + & 2x_2 & = & 1 \\ 2x_1 & + & 4x_2 & = & 2 \\ -x_1 & - & 2x_2 & = & -1 \\ -2x_1 & - & 4x_2 & = & -2 \end{array} \end{array}$$

Observe que $m > n$, mas $rank(A) = rank([A|b]) = 1 < n$ pois a segunda, a terceira e a quarta linha são múltiplos da primeira linha, contradizendo a afirmação do Prof. Kunz.

EX 18.5: Note que minimizar uma função $f(x)$ equivale a minimizar $2f(x)$. Logo: min $q(x) \Leftrightarrow$ min $2q(x) \Leftrightarrow$ min $\frac{2}{2}x^TQx + 2d^Tx + 2c \Leftrightarrow$ min $\frac{1}{2}x^TQx + \frac{1}{2}x^TQ^Tx + 2d^T + 2c \Leftrightarrow$ min $+\frac{1}{2}x^T(Q+Q^T)x + 2d^Tx + 2c$. Assim podemos resolver o problema em consideração resolvendo um problema tipo P_Q onde Q é substituída por $Q + Q^T$, d por $2d$ e c por $2c$. Então resolvemos o problema abaixo:

$$\begin{array}{ll} \text{Minimize} & \frac{1}{2}x^T(Q+Q^T)x + 2d^Tx + 2c \\ \text{Subjeito a :} & \\ & Ax \geq b \end{array}$$

EX 18.6: Obtenha primeiramente o Lagrangeano:

$$\mathcal{L}(w, b, e, \lambda) = \frac{1}{2}w^Tw + \frac{c}{2}\sum_{i=1}^{m} e_i^2 + \sum_{i=1}^{m} \lambda_i(y_i - w^T\phi(x_i) - b - e_i)$$

Agora, calcule as derivadas parciais:

$$\partial\mathcal{L}/\partial w = w - \sum_{i=1}^{m} \lambda_i\phi(x_i) \tag{A.19}$$

$$\partial\mathcal{L}/\partial b = -\sum_{i=1}^{m} \lambda_i \tag{A.20}$$

$$\partial\mathcal{L}/\partial e_i = ce_i - \lambda_i,\ i = 1, \ldots, m \tag{A.21}$$

$$\partial\mathcal{L}/\partial \lambda_i = y_i - w^T\phi(x_i) - e_i,\ i = 1, \ldots, m \tag{A.22}$$

Uma vez que as derivadas de $\mathcal{L}$ devem ser nulas, para satisfazer condições necessárias de otimalidade, fazendo:

$$e_i = \lambda_i/c,\ i = 1, \ldots, m \tag{A.23}$$

$$w = \sum_{i=1}^{m} \lambda_i\phi(x_i) \tag{A.24}$$

e substituindo em (A.22), obtemos as equações (A.19)–(A.22) do Lagrangeano em função das variáveis λ_i e b:

$$\sum_{i=1}^{m} \lambda_i = 0 \tag{A.25}$$

$$y_i - \sum_{j=1}^{m} \phi(x_j)^T\lambda_j\phi(x_i) - b - \lambda_i/c = 0,\ i = 1, \ldots, m \tag{A.26}$$

De forma mais compacta, (A.25)–(A.26) podem ser colocadas em notação matricial:

$$\begin{bmatrix} 0 & \mathbf{1}^T \\ \mathbf{1} & Q + c^{-1}I \end{bmatrix} \begin{bmatrix} b \\ \lambda \end{bmatrix} = \begin{bmatrix} 0 \\ y \end{bmatrix} \tag{A.27}$$

onde: $\mathbf{1} = (1, \ldots, 1)^T$, $y = (y_1, \ldots, y_m)^T$, $\lambda = (\lambda_1, \ldots, \lambda_m)^T$ e Q é uma matriz $m \times m$ com entrada $Q_{ij} = \phi(x_i)^T\phi(x_j)$. A solução do problema P_Q se reduz a resolver a equação (A.27) e encontrar w a partir de (A.24), assim obtendo o aproximador ótimo $\hat{f}(x, w, b) = w^T\phi(x) + b$.

EX 18.7: Basta fazer $q = -bA^{-1}u$.

A.19 Programação Não-Linear sob Restrições: Algoritmos

EX 19.1: Os códigos Matlab da função barreira e do algoritmo de barreiras seguem abaixo. A Figura A.16 dá a trajetória gerada pelo algoritmo de barreiras. A seqüência de iterandos é convergente para a solução ótima $(x_1^*, x_2^*) = (4, 8)$.

```
%-----------------------------
% bm - barrier method

x = [];
x(1,1) = 0.5; x(1,2) = 0.5;
mu = []; mu(1) = 100;

opt = optimset('fminsearch'); opt = optimset(opt,'MaxIter',100);
opt = optimset(opt,'TolX',1e-8);

for k=2:20
   xi = x(k-1,:)';
   z = fminsearch('b',xi,opt,mu(k-1));
   x(k,1) = z(1);
   x(k,2) = z(2);
   mu(k) = mu(k-1)*0.4;
end
```

```
%-----------------------------
% b(x; mu)
function r = b(x, mu)

x1 = x(1); x2 = x(2);
r = -3*x1 - 2*x2 ...
   - mu*log(    1 - x1  +  2*x2) ...
   - mu*log( 2 - x1  +  x2)  ...
   - mu*log( 6 - 2*x1 + x2) ...
   - mu*log( 5 -   x1) ...
   - mu*log( 16 - 2*x1 - x2) ...
   - mu*log( 12 - x1 - x2) ...
   - mu*log( 21 - x1 - 2*x2) ...
   - mu*log( 10 - x2) ...
   - mu*log( x1 ) ...
   - mu*log( x2 );
```

EX 19.2: Conversões entre algumas representações são dadas abaixo.

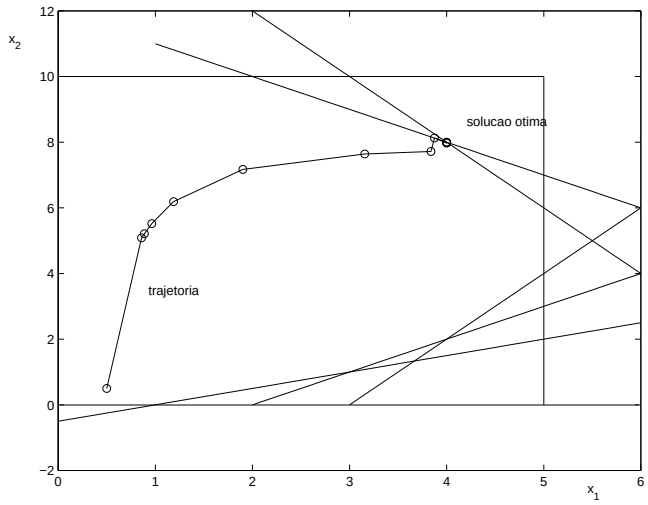


Figura A.16: Trajetória de iterandos produzida pelo algoritmo de barreiras.

$1 \leftrightarrow 2$**:** A conversão pode ser demonstrada como segue:

$$\begin{aligned}
\{x : (x-x_0)^T A^{-1}(x-x_0) \leq 1\} &\Leftrightarrow \{x : \|A^{-\frac{1}{2}}(x-x_0)\|^2 \leq 1 \\
&\Leftrightarrow \{x : \|A^{-\frac{1}{2}}(x-x_0)\| \leq 1 \\
&\Leftrightarrow \{x : \|u\| \leq 1\} \text{ com } u = A^{-\frac{1}{2}}(x-x_0) \\
&\Leftrightarrow \{x : \|u\| \leq 1\} \text{ com } A^{\frac{1}{2}}u + x_0 = x \\
&\Leftrightarrow \{A^{\frac{1}{2}}u + x_0 : \|u\| \leq 1\} \\
&\Leftrightarrow \{Bu + x_0 : \|u\| \leq 1\} \text{ com } B = A^{\frac{1}{2}}
\end{aligned}$$

Note que $det(A)^{\frac{1}{2}} = det(A^{\frac{1}{2}}) = det(B)$.

$2 \leftrightarrow 3$**:** Uma vez que $x \in \mathcal{E}$ na representação 2 se $x = Bu + x_0$, temos que $u = B^{-1}(x - x_0) = Ax - b$ com $A = B^{-1}$ e $b = B^{-1}x_0$. Observe que $Bu + x_0 = B(Ax - b) + x_0 = x - Bb + x_0 = x - BB^{-1}x_0 + x_0 = x$. Portanto $\mathcal{E} = \{Bu + x_0 : \|u\| \leq 1\} = \{x : \|Ax - b\| \leq 1\}$. Note que $det(B) = det(A^{-1})$.

$3 \leftrightarrow 4$**:** A conversão pode ser desenvolvida como segue. $\|Ax - b\| \leq 1 \Leftrightarrow (Ax - b)^T(Ax - b) \leq 1 \Leftrightarrow x^T A^T Ax - 2b^T Ax + b^T b = x^T Cx + 2d^T x + e$ para $C = A^T A = A^2$, $d = -A^T b$, e $e = b^T b$.